LA PRIMERA REVISTA DE PROGRAMACION EN CASTELLANO

SÓLO PROGRAMADORES

Revista especializada para usuarios de PC

AÑO 3. Nº 18
250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $

ESCRITA por desarrolladores para desarrolladores

Linux: Estructura del Núcleo

Creación de un compilador: Analizadores Sintácticos

Ray Tracing: TEXTURAS

Lenguaje HTML Documentos profesionales

Y además:

- DEMOS: Los Sprites
- UNIX: Sockets
- PROGRAMACIÓN: Recursividad
- SONIDO: Las Demos
- HIPERTEXTO: Creación de DLLS

REGALO CD-ROM LINUX SLACKWARE 3.00

CURSO DE PROGRAMACIÓN VISUAL C++ 4.0

TOWER COMMUNICATIONS, S.R.L.

413042 303299 000018

XXI JORNADAS DE COMUNICACIONES, INFORMATICA Y ELECTRONICA

96

4-7 DE MARZO

ETSI DE TELECOMUNICACION • CIUDAD UNIVERSITARIA. MADRID

UNIVERSIDAD POLITECNICA DE MADRID ■ COLABORA: FUNDACION UNIVERSIDAD-EMPRESA

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

FEBRERO 1996. Número 18
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló
Director Editorial
Mario de Luis García
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador técnico
Eduardo Toribio Pazos
Publicidad
Mª Eugenia González
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José María Peco, José Carlos Remiro, Enrique de Alarcón, Agustín Guillén, Héctor Martínez, Álvaro Silgado, David Aparicio, Ignacio Cea, Enrique Castañón, Daniel Navarro, Jorge del Río.

Maquetación
Fernando García Santamaría
Tratamiento de imagen
María Arce Giménez
Servicio Técnico
Óscar Rodríguez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Ilustraciones
Miguel Alcón
Secretaría de Redacción
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Marqués de Portugalete, 10
28027 MADRID
Telf.: 741 26 62 / Fax: 320 60 72
Filmación
Filma Dos S.L.
Impresión
Gráficas Reunidas, S. A. Madrid
Distribución
SGEL

La revista SÓLO PROGRAMADORES no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados.
El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792

CARTA DEL DIRECTOR

EL CAMALEÓN AGAZAPADO

El otro día la editorial mantuvo una reunión con una importante compañía de distribución de software. En el transcurso de la misma su director me preguntó: "por cierto, ¿quién es ese que hace las *CARTAS DEL ILUSO* en SÓLO PROGRAMADORES, un programador, un periodista, acaso *tú mismo*?.-No, le contesté. Es una larga historia que creo merece la pena compartir con todos vosotros.

Hace mucho tiempo conocí, a través de otros programadores a un tipo alocado y genial que se dedicaba a hacer música utilizando el ordenador. Por aquel entonces y por motivos que no vienen al caso, yo necesitaba un músico. Bingo, dí con la persona correcta. Aunque un poco extravagante, nuestro *iluso* resultó ser además de un excelente programador y diseñador gráfico, además de un tipo divertido amigo de sus amigos. El tiempo pasó y nuestros caminos, como a veces pasa en la vida, se separaron. Cinco años después surgió la idea de hacer SÓLO PROGRAMADORES y yo eché mano de la agenda en la que, año tras año, fui registrando todos los personajes singulares con los que topé en los comienzos de mi andadura profesional. Y apareció él. Confieso que sólo su nombre y recuerdo hace que se dibuje una sonrisa en mi cara. Quedamos. "¿Cómo te va la vida?" -Bien, trabajo en un gran banco en el departamento de sistemas informáticos. "Y la carrera, ¿la acabaste?" -la verdad es que no, me puse a trabajar y ya sabes lo que ocurre: fui dejándolo poco a poco... Era una historia que sonaba familiar y muy cercana. Mi amigo empezó a escribir para SP con las columnas que todos conocemos y, pasados unos meses, contrajo matrimonio con su novia *de toda la vida*. Me alegré por él. Desde entonces, el trabajo diario, sus obligaciones y los compromisos hicieron que la columna de SP la hiciese (según sus propias palabras) "más que por dinero, por el compromiso personal con un amigo". Supongo que algo parecido le debe haber ocurrido con el resto de las cosas que no pertenecen a su trabajo. Ahora las hace por vocación. Como otros tantos genios creativos su talento ha quedado encerrado bajo la necesidad de vivir día a día. Sin embargo y pese a las limitaciones de tiempo, es difícil que cuando una persona lleva tantas cosas dentro no las saque al exterior aunque sea sin darse cuenta. Y ese es el caso de nuestro amigo *iluso*, quien con sus columnas expresa el día a día de muchos de nosotros.

Entiendo que el directivo de la empresa de software de la que hablaba *al principio* es uno de los nuestros. En el fondo, siente la necesidad de crear algo y la informática es la herramienta por la que se ha decantado. Es cierto que muchos de nosotros podemos hacer otras cosas además de programar: coordinar, evaluar, pero en el fondo, todos llevamos un *iluso* dentro, alguien que por encima de todo, quiere crear. A ese personaje, el *camaleón agazapado* que hay en cada empresa, va dedicado nuestro editorial.

Y hablando de este número, que es la obligación, entregamos la versión 3.00 del Linux Slackware. En el CD también hay una demo de las herramientas de programación de Microsoft, aprovechando que en este número se inicia un curso de programación en Visual C++. Por último, recordar que el concurso de *demos* esta cerrado y en el próximo número conoceréis los resultados. Hasta el mes que viene.

MARIO DE LUIS

SUMARIO

NÚMERO 18

6 **CONTENIDO CD-ROM**
Ante la intensa demanda de nuestros lectores un nuevo Slackware de Linux, la última y novedosa 3.0. Además Lotus Notes 4, el estándar del groupware.

8 **NOTICIAS**
El espacio dedicado a informar al lector de las últimas novedades en el mundo de la informática y el comentario de los libros de interés.

12 **WORLD WIDE WEB**
Tercera entrega del curso de programación en lenguaje HTML. Se pretende que el lector sea capaz de crear sus propias páginas WEB.

18 **CURSO DE PROGRAMACIÓN**
Un nuevo capítulo de esta entrega orientada a aquellos lectores con un nivel más bajo de conocimientos en programación y a los que estén comenzando. Este mes la recursividad.

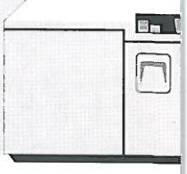
24 **CURSO DE UNIX**
El mes pasado se abrió la puerta de las comunicaciones UNIX mediante la interfaz socket. En el presente número se verán algunos ejemplos para repasar conceptos.

30 **CÓMO HACER UNA DEMO**
El manejo de los sprites, de forma original, da paso a sorprendentes posibilidades que complementan una demo. Este mes cómo usar un sprite.

34 **TÉCNICAS DE SONIDO**
Cómo sincronizar la música con el programa que está corriendo en primer plano es el tema que se abarca en el nuevo capítulo de la serie dedicada a técnicas de sonido.

37 **CURSO DE RAY TRACING**
Con el uso de las texturas es posible crear objetos de una complejidad sorprendente con muy poco esfuerzo. Es un nuevo paso hacia la reproducción de la realidad.

42 **GRANDES SISTEMAS**
La concurrencia entre aplicaciones ejecutadas en una máquina monoprocesador es uno de los grandes problemas por resolver por parte de cualquier sistema operativo.

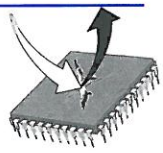


47 SISTEMA OPERATIVO LINUX
Este mes en la sección de Linux se abarca el tema de la organización del núcleo principal de este sistema operativo, y sus principales componentes.

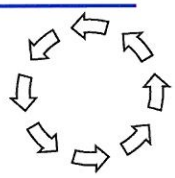
51 PROGRAMACIÓN EN CLIPPER
Clipper es uno de los lenguajes de programación más solicitados profesionalmente hoy en día dentro del campo de las bases de datos.

56 PROGRAMACIÓN EN C++
En esta nueva entrega ha llegado la hora de comenzar a integrar todos los conocimientos adquiridos en capítulos anteriores en clases C++.

61 CREACIÓN DE UN COMPILADOR
Se estudiarán las reglas sintácticas de un lenguaje de programación, para pasar a implementarlas en un nuevo elemento de todo compilador; el analizador sintáctico.



67 CURSO DE HIPERTEXTO
Última entrega de la serie dedicada a las técnicas de construcción de hipertexto, se aborda la creación de librerías de enlace dinámico para el sistema de ayuda de Windows.

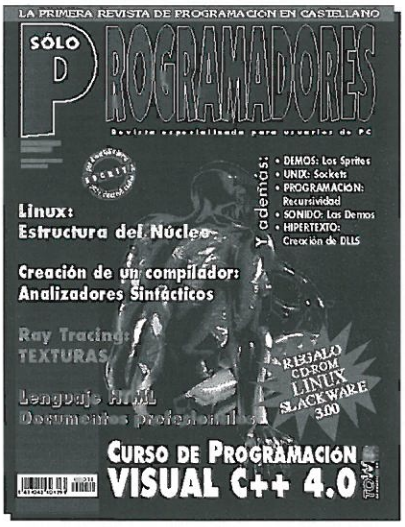


71 PROGRAMACIÓN EN DPMI ABAJO NIVEL
Hoy en día todo el mundo usa extensores del DOS para poder crear programas más potentes. Estudiemos el más estándar del mercado: el DPMI de la casa Microsoft.

74 CURSO DE VISUAL C++
Este mes comienza lo que será uno de los platos fuertes todos los meses en Sólo Programadores: Visual C++ 4.0 todo un entorno de desarrollo.

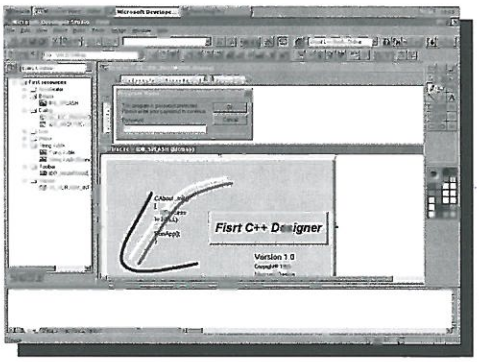
81 CORREO DEL LECTOR
Es el espacio dedicado a la resolución de los problemas surgidos a nuestros lectores en los diversos aspectos de la programación.

DESTACADO



Este mes en Sólo Programadores y a consecuencia de la gran demanda por parte de nuestros lectores de regalo un CD-ROM con el Slackware 3.00 de LINUX.
Además todas las secciones habituales de cada mes.

VISUAL C++ 4.0



Sólo Programadores este mes comienza un nuevo curso dedicado a una de las herramientas de programación más potentes del mercado: Visual C++ 4.0, todo un entorno de desarrollo

CONTENIDO DEL CD-ROM

LINUX SLACKWARE 3.00

Este mes ofrecemos, en nuestro CD-ROM de regalo, la última versión de la distribución Linux SlackWare. Se trata de un cambio importante con respecto a anteriores versiones, puesto que se trata de la primera basada en el formato ELF. Este formato es el que se ha adoptado como estándar en Linux a partir de esta versión. Además de la mejora de rendimiento, este formato proporcionará mayor compatibilidad con el resto de entornos Unix para Intel. Los programas de las versiones anteriores de Linux siguen estando soportados, pero se requerirá una mayor cantidad de memoria si deseamos ejecutar simultáneamente programas de ambo tipos. . Por el momento, pasaremos a la instalación del CD-ROM.

En primer lugar, deberemos disponer de un disco con 100 o más megabytes disponibles para nuevas particiones. Para ello, si el disco contiene ya otros sistemas operativos, se necesitará reducir el tamaño de las particiones antiguas, haciendo un backup previo y restaurando los datos en el nuevo espacio. Esta operación es tediosa, y puede automatizarse (sin necesidad de un backup total) mediante FIPS, un programa incluido en el CD, en *\linux300\install\fips.exe*.

Si ya tenemos Linux en nuestro PC, y deseamos instalar esta versión sobre la anterior, no necesitaremos reparticionar nada. Recuerde salvar los datos de interés (configuración de X-Windows, por ejemplo), antes de realizar la nueva instalación. Dado que se trata de un cambio mayor entre ésta y la anterior versión de SlackWare, es preferible realizar una instalación total a sustituir sólo algunos ficheros. Además, los programas "extra" que haya incorporado a su sistema deberan ser recompilandos de nuevo si no se encuentran en la nueva distribución, para mejorar su rendimiento. Si dispone de un disco suficientemente grande, también puede instalar SlackWare 3.00 sin borrar la versión anterior de Linux, y comparar el rendimiento de ambas para decidir si le interesa la actualización.

Una vez que se dispone de espacio en disco duro, elegiremos un disquete de arranque (*boot*) y otro de setup (*root*) para proceder a la instalación. El disco de arranque depende del hardware del PC que utilicemos. Será suficiente elegir un disco que reconozca nuestro lector de CD-ROM y el disco duro destino. Tras concluir la instalación y rearrancar, podremos recompilar el núcleo para explotar completamente nuestro hardware, o elegir uno de los que se proporcionan en la serie Q.

A grandes rasgos, para sistemas SCSI donde no necesitemos soporte de red durante la instalación, el disco boot elegido será *scsi.gz*. Si nuestro sistema no es SCSI, y el lector de CD es IDE/ATAPI, deberemos tomar idecd.gz. Los lectores de sistema propietario, normalmente de doble velocidad, necesitarán *sbpcd.gz* (Creative/Panasonic), *cdu31a.gz* (Sony) ó *mitsumi.gz* (Mitsumi) respectivamente. No deberemos confundir el lector Mitsumi antiguo, enganchado al conector de la tarjeta de sonido o a una placa especial, con el Mitsumi IDE 3x ó 4x, que se conecta a la controladora del disco duro. El primero usará el disco *mitsumi.gz*, y el segundo el *idecd.gz*. Lo mismo reza para las nuevas unidades de Creative, y, en general, cualquier modelo que haya cambiado de un bus propietario al bus IDE.

Si ninguno de los discos de arranque reconoce nuestro lector, siempre es posible copiar parte del CD al disco duro con MS-DOS, elegir como arranque el fichero *bare.gz*, e instalar desde el propio disco. Con disponer de las series A, D y Q, será suficiente para una instalación inicial, que luego permitirá recompilar un núcleo que reconozca nuestro lector. En el remoto caso de que sigamos sin acceder al CD, es posible que todavía no esté soportado, y deberá esperar a una versión más moderna de Linux, o que la configuración de nuestro sistema no se reconozca automáticamente, con lo que deberemos leer la documentación que viene con la distribución.

Una vez escogido el disco de arranque, el disco de setup suele ser, para casi todos los PC de sobremesa, el *color.gz*. Los poseedores de portátiles también pueden probar el nuevo *pcmcia.gz*. Si no funcionan, podemos elegir *text.gz*, el disco mas básico para la instalación.

En este punto, necesitamos dos discos con formato, pero en blanco. Supuesto que los discos de arranque y setup son *bare.gz* y *text.gz*, que nuestro disco duro es C:, y que el lector de CD se encuentra en D:, la secuencia de creación de los discos será la siguiente:

```
C:> MKDIR TMP
C:> CD TMP
C:> CD D:\LINUX300\BOOTDISK
C:> COPY D:BARE.GZ
C:> CD D:\LINUX300\ROOTDISK
C:> COPY D:TEXT.GZ
C:> CD D:\LINUX300\INSTALL
C:> D:GZIP -d BARE.GZ
C:> D:RAWRITE
(introduzca un disco en blanco en A:)
(el fichero origen es BARE)
(la unidad destino es A:)
C:> D:RAWRITE
(introduzca otro disco en A:)
(el fichero origen es TEXT.GZ)
(la unidad destino es A:)
C:> CD\
C:> DELTREE TMP
```

Con esta operaciones, habremos conseguido los dos discos necesarios para comenzar la instalación. Los lectores que hayan instalado Linux con anterioridad, notarán una diferencia importante. En esta versión SlackWare, el disco de setup se crea a partir de un fichero sin descom-

primir. Esto es necesario debido a que el tamaño del mismo ya supera, en algunos casos, la capacidad del disquete.
Tras este paso, debemos rearrancar el PC con el disco de arranque dentro de la unidad A:. Aparecerá un mensaje de bienvenida, y el prompt

boot:

Pulsaremos return y esperaremos que el arranque se complete. Si tuviésemos una configuración especial que Linux no detectase, habríamos tenido que introducir parámetros especiales antes de pulsar la tecla de retorno. Si llega el caso, deberá consultar la documentación para resolver su caso concreto.

Ahora se nos mostrarán mensajes que nos informan del hardware que se está detectando. Deberá aparecer el disco duro donde realizar la instalación, y el lector de CD a partir del que realizarla. Para las personas que realicen este paso por primera vez, vamos a introducirles un poco en la nomenclatura que sigue Linux para reconocer las unidades. Suponiendo un sistema IDE con dos interfaces (dos cables y dos unidades en cada cable), la unidad 'master' del primer cable es *hda*, la 'slave' es *hdb*, la 'master' del segundo cables es *hdc* y la 'slave' es la *hdd*. Para SCSI, las unidades del bus (de 0 a 7) se nombran *sda* hasta *sdh*, respectivamente. Las particiones dentro de cualquier unidad van numeradas. Por ejemplo, la segunda partición del disco duro IDE de arranque es *hda2*.

Cuando finalicen los mensajes propios del primer disco, se producirá una pausa y se nos pedirá que introduzcamos el disco RAM. El segundo disco que hemos preparado, el de setup, es el que deberemos insertar ahora. Tras pulsar intro, Linux descomprime el disco y lo copia en una imagen de RAM, con el objetivo de acelerar la ejecución de la instalación posterior.

Una vez que se haya completado la carga, se muestra el login del sistema. Podemos entrar tecleando "root".
En primer lugar, deberemos crear al menos dos particiones, una de swap y otra de sistema, en el espacio libre que habremos reservado con anterioridad, mediante FIPS o cualquier otro método. La primera partición es una zona que se suma a la RAM del sistema, permitiendo que el sistema aumente su capacidad para ejecutar programas. La segunda es donde se almacena la estructura de directorios y ficheros que forman Linux. El programa particionador se invoca con

fdisk

Se basa en comandos de una letra. Con 'm' tendremos un resumen de todos los comandos disponibles. En resumen, disponemos de los siguientes:
'n' para crear una partición
't' para cambiar su tipo
'd' para borrarla
'p' para mostrar la tabla actual
'a' marca la particion de arranque,
'w' para grabar y salir, y
'q' para abandonar sin grabar.

Los usuarios de OS/2 deberán crear las particiones desde el fdisk de dicho sistema, y luego cambiar los tipos de las mismas desde el fdisk de Linux. Dependiendo del disco disponible, y si desea poder recompilar cualquier programa, cree una partición de swap (tipo 82 hexadecimal) de forma que la suma de RAM y swap sea de 16 Mb como mínimo, y de 32 Mb o más como tamaño recomendado. La partición de sistema (tipo 83 hexadecimal) debe ser de 40 Mb como mínimo, aconsejándose 100Mb o más.

Si disponemos de menos de 8 Mbytes de RAM, habrá que activar el swap a mano. Supongamos que tenemos un swap de 8192 sectores en la segunda partición del primer disco IDE, se formateará y activará el área de intercambio mediante:
mkswap /dev/hda2 8192
swapon /dev/hda2
Al teclear esto, tenga en cuenta que cuando setup le consulte mas tarde sobre si ejecuta 'mkswap' o 'swapon', deberá responder que no a ambas cuestiones.
Si por el contrario ya dispone en su PC de RAM suficiente, no necesitará teclear estos comandos. A partir de aquí, en cualquier caso, dispondremos de más de 8 Mb de espacio de memoria total, imprescindibles para ejecutar el siguiente comando. Teclee

setup

Con el disco de setup a color, dispondrá de un vistoso menú donde se le guiará paso a paso por el resto de la instalación. Deberemos puntualizar varios aspectos que se le presentarán a lo largo de la ejecución de este programa, que por otra parte es muy intuitivo si se disponen de unos mínimos conocimientos de inglés.

Cuando se nos pregunte por el tipo de teclado, se nos pedirá escoger entre una lista de ficheros, donde buscaremos la opción 'es.map.gz' (séptima línea).

El origen de la instalación normal desde CD-Rom será el directorio /linux300/slakware.

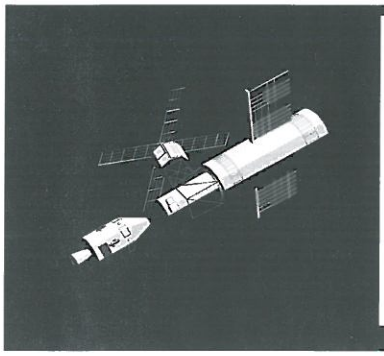
Los grupos de discos de instalación se denominan series, y agrupan los programas según su temática. Son imprescindibles para sacar algún partido a Linux, tanto la serie A (básica) como la D (desarrollo). El resto son opcionales y los instalaremos conforme a nuestros intereses.

Los tagfiles son los descriptores que indican al programa de instalación sobre los ficheros que son imprescindibles u opcionales.

La zona horaria que corresponde a España es MET, y debe seleccionarse la misma cuando setup nos lo pida.

La selección de los ficheros que se instalan en cada serie ha mejorado, puesto que es ahora a pantalla completa, en lugar de preguntar por el destino de cada fichero como en anteriores versiones.

Por último, existen varias formas de arrancar con Linux. Una de ellas es mediante un disco de arranque, que será creado casi en los últimos pasos de la instalación.
El segundo método es mediante LILO, un cargador que se graba en el sector de arranque de la partición de Linux (o en la MBR del disco), y permite elegir entre varias particiones y sistemas en el momento de iniciar el sistema. Es similar al Boot Manager de OS/2, y se instala si así lo decidimos en el menú de selección de ficheros de la serie A.
Por último, LOADLIN es un ejecutable MSDOS que permite arrancar desde éste sistema operativo, pasándo como parámetro el núcleo de Linux a arrancar, y la partición que contiene el sistema Linux. Se instala cuando se selecciona en la misma pantalla que LILO, y deposita un fichero en */root* que deberá ser descomprimido para leer documentación detallada sobre su instalación final.
Una vez que salga del setup, rearranque su sistema para comprobar si todos funciona correctamente, y lea la documentación adjunta para afinar los aspectos que desee de su sistema.



SÓLO PROGRAMADORES

NOTICIAS

Bay Networks anuncia BayStack

Bay Networks ha anunciado recientemente una nueva solución apilable de routers, hubs, conmutadores y capacidades de gestión integrada de red denominada BayStack. Esta nueva solución proporciona a los entornos de grupos de trabajo una línea de productos apilables de primera clase diseñados para ser administrados e integrados como una única solución completa.

La familia de productos está formada por el BayStack 10BASE-T Ethernet Stackable Hub y del nuevo software de gestión de red Optivity Workgroup. También se incluyen versiones del Ethernet Workgroup Switch, el 100BASE-T Hub y los nuevos routers de acceso remoto Access Node (AN) y Access Node Hub (ANH).

El nuevo software de gestión de red Optivity Workgroup proporciona herramientas para gestionar, configurar y eliminar los problemas de hubs, routers y conmutadores, entre otros componentes de red. Consta de las aplicaciones EZ LAN y EZ Internetwork.

BayStack puede configurarse para 8 usuarios y se puede actualizar con facilidad para soportar centenares de usuarios.

Para más información:
Bay Networks
Ricardo Miranda-Naón
Tel : (91) 383 13 20

Intel duplica las prestaciones del controlador MCS 96

Intel Corporation ha anunciado un nuevo miembro de la familia de microcontroladores MCS 96 desarrollado para el control de transmisión y del motor. El controlador 83C196EA opera a 32 MHz, lo que supone casi dos veces la velocidad de trabajo de los dispositivos existentes a 16 MHz. Además, el 83C196EA incorpora una nueva SDU (Serial Debug Unit) para ayudar a los desarrolladores de software a refinar el código y los parámetros. También incluye un interfaz de memoria integrado en el chip para unos diseños de sistemas compactos.

El nuevo controlador dispone de 17 canales flexibles EPA (Event Processor Array) y de 8 módulos sólo de salida para proporcionar un mayor control. Sus canales de conversor 16 A/D permiten la utilización de más sensores para determinar el estado operativo del motor. El 83C196EA también incorpora un controlador de bus diseñado para trabajar con dispositivos de memoria flash de bajo coste. Este interfaz de controlador a memoria flash optimiza las prestaciones del 83C196EA reduciendo al mismo tiempo los costes de la tarjeta y el espacio.

La producción en volumen del 83C196EA está prevista para Marzo de 1996, aunque ya existen muestras en un encapsulado QFP de 160 pines. El precio recomendado en los Estados Unidos es de 27 dólares por 1.000 unidades.

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web: http://www.intel.com

Nuevas versiones de SQL Serve Monitor y SQL Server Manager

Sybase Inc. ha anunciado nuevas versiones de sus herramientas de gestión y monitorización del rendimiento de las bases de datos como parte integrante de su Sistema de Gestión de Bases de Datos Relacionales (SGBDR) SQL Server 11.

SQL Server Monitor es una herramienta de fácil utilización que permite a los administradores del sistema ver los datos de rendimiento en tiempo real y las tendencias históricas. Las características del nuevo SQL Server Monitor permiten, entre otras acciones:

- Controlar los sistemas SQL Server sin afectar al rendimiento.
- Identificar fácilmente las tablas de acceso, los procedimientos de almacenamiento y otras estadísticas.
- Optimizar el rendimiento del Gestor de Memoria Lógica de SQL Server 11.
- Visualizar gráficamente información sobre los servidores, dispositivos, bases de datos, etc.

SQL Server Manager es una herramienta basada en Windows que ofrece todas las capacidades operativas necesarias para la administración local y de los servidores remotos. Esta versión soporta todas las nuevas características del SGBDR Sybase SQL Server 11 y ofrece las siguientes mejoras: gestión de las cachés designadas, parámetros de configuración de System 11 para un completo soporte y partición de las tablas.

Para más información:
Agencia S.E.I.S.
Ludi García / Tony Noel
Tel : (91) 308 43 15
Fax : (91) 310 38 76

Novell lanza TUXEDO System 6.1

Novel Inc. ha anunciado la disponibilidad de TUXEDO System 6.1, la última generación de control abierto de transacciones. Este producto incluye una serie de nuevas características y mejoras que proporcionan al usuario una mayor facilidad de desarrollo, fiabilidad y rentabilidad en las aplicaciones de misión crítica distribuidas a lo largo de redes cliente/servidor.

TUXEDO System 6.1 se ejecuta a lo largo de múltiples plataformas y soporta APIs abiertos para ayudar a los clientes a realizar con facilidad la transición a redes cliente/servidor. Está concebido para soportar aplicaciones críticas de fabricación, servicios de información en línea, gestión de venta al por menor, finanzas, nuevas formas de comercio electrónico y otros sectores. Entre sus características destacan:

- Gestión mejorada de aplicaciones a través de una nueva interfaz visual de usuario para configurar, controlar y gestionar las aplicaciones distribuidas.
- Potentes capacidades de comunicación para aplicaciones de misión crítica a través de nuevos servicios "Event Broker" basados en el paradigma Publicación/Suscripción.

TUXEDO System 6.1 ya se encuentra disponible a través de Novell y sus servicios OEM. La versión runtime comienza por un PVP aconsejado de 395 dólares por usuario, mientras que la versión de desarrollador comienza por el PVP aconsejado de 2.395 dólares para un sólo usuario.

Para más información:
Novell Spain
Daniel Toledano
Tel : (91) 577 49 41

Bay Networks refuerza su oferta ATM

Bay Networks, Inc. ha anunciado sus alianzas estratégicas con Interphase Corporation y ADC Kentrox para proporcionar soluciones backbone ATM LAN y WAN transparentes. Según los términos de estos acuerdos, Interphase y ADC Kentrox proporcionarán a los clientes de Bay Networks adaptadores de red ATM y concentradores de acceso ATM respectivamente. Se encuentran disponibles inmediatamente backbones LAN basados en LAN Emulation (LANE), con soporte futuro para el I-PNNI (Integrated Private Network to Network Interface) y el servicio de datos ABR (Available Bit Rate) definido por el ATM Forum.

En el momento que las especificaciones I-PNNI y ABR del ATM Forum estén terminadas las tres compañías trabajarán conjuntamente para entregar el mismo nivel de interoperabilidad y facilidad de uso que han proporcionado con éxito en las redes actuales basadas en LANE. I-PNNI es una próxima generación de protocolo de rutado que opera a través de interconexiones de redes que incluyen conmutadores ATM, servidores conectados ATM y conmutadores de tramas (conmutadores "edge"), así como routers que no están conectados directamente a una red ATM. I-PNNI permite el rutado a través de redes dispares basadas en tramas y celdas, y proporciona soporte para QoS (Quality of Service). ABR especifica los métodos para manejar la congestión en redes ATM.

Para más información:
Abanico/Hot Line
Liliane Chinyavong
Tel : (91) 383 13 20

Soluciones de almacenamiento masivo automatizado

Memorex Telex anunció recientemente unas nuevas soluciones de almacenamiento masivo automatizado que proporcionan una mejor fiabilidad y un mayor ahorro de espacio, además de capacidades de robótica y prestaciones de montaje de cartucho más rápido para los clientes que tienen entornos de centro de datos con plataformas mixtas. Los nuevos productos de Memorex Telex ofrecen a los clientes soluciones que satisfacen las necesidades actuales de almacenamiento de empresa, como la conexión multiplataforma a mainframes, servidores UNIX, AS/400 y servidores PC LAN. Estas soluciones ofrecen los beneficios de la seguridad e integridad de los datos del mainframe a los usuarios de entornos de red compartidos.

La nueva ATL (Automated Tape Library) 5.600, la unidad de cinta 5.498 mejorada y el controlador 5.499 de Memorex Telex , junto con el LMS (Library Management Software) exclusivo de Memorex Telex proporcionan una solución de cinta automatizada que amplía las ofertas de almacenamiento masivo de la empresa.

Para más información:
Memorex Telex
Carlo d'Amore
Tel : (91) 372 61 57
World Wide Web: http://www.mtc.com

Bentley anuncia MicroStati on Modeler II

Bentley Systems, Inc. ha anunciado una versión actualizada del modelador de sólidos para diseño mecánico, MicroStation Modeler II. La nueva versión incluye ahora herramientas de creación de geometría avanzadas y un diseño unificado de sólido/superficie. Está optimizado para la "Class 95", lo que significa que ahora saca partido del nuevo entorno MicroStation 95 de alta productividad.

MicroStation Modeler II, basado en ACIS, ha sido actualizado para soportar ACIS versión 1.7, la más reciente versión de la tecnología de modelado de sólidos Spacial Technology. El producto ahora soporta también OpenGL, y dispone de la tecnología de dibujo AccuDraw y de MicroStation Basic.

Para más información :
Bentley Systems
Jesús Pazos
Tel : (91) 372 89 75
World Wide Web:
http://www.bentley.com

NOVEDADES

Intel y Novell lanzan ManageWise 2.0

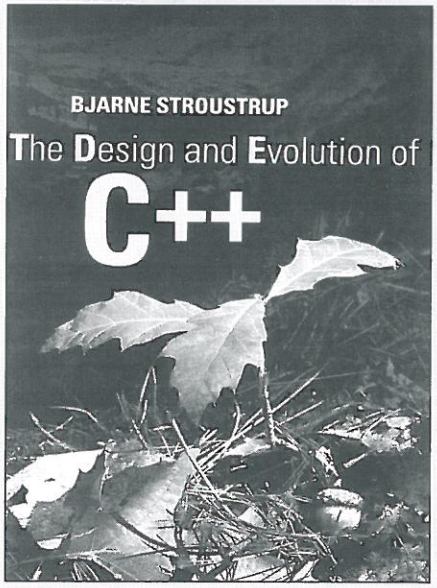
Novell, Inc. e Intel Corporation han anunciado las primeras ventas a usuarios del ManageWise 2.0, la nueva versión de la principal solución integrada de gestión de servidor, ordenador de sobremesa e infraestructura de todas las redes Novell. Esta versión ofrece la primera integración real de gestión de redes y sistemas para PC LAN y permite que los administradores se muevan de forma homogénea entre la gestión del Protocolo de Gestión de Red Simple (SNMP) y la administración del Directorio de Servicios NetWare (NDS) de la red.

ManageWise 2.0 integra sus capacidades dentro de la red en lugar de ser un sistema centralizado. Reduce el tráfico de la red y proporciona un acceso compartido a la información de gestión, aprovechando la capacidad de procesamiento de la red. Además, distribuye direcciones, umbrales y control de dispositivos críticos. ManageWise 2.0 se ha constituido como la primera solución de gestión que utiliza la capacidad de NDS como una estructura de seguridad para la gestión SNMF, y permite que los empleados IS identifiquen a los administradores de la red y regulen su acceso a las funciones de gestión.

Creative Labs

3D Blaster es la primera tarjeta de juego de alto rendimiento disponible para usuarios. Es el primer producto de la familia Blaster en proporcionar gráficos mapeados de textura real en tres dimensiones dentro del ordenador domestico. Se comercializará en España en éste primer trimestre del 96.

LIBROS



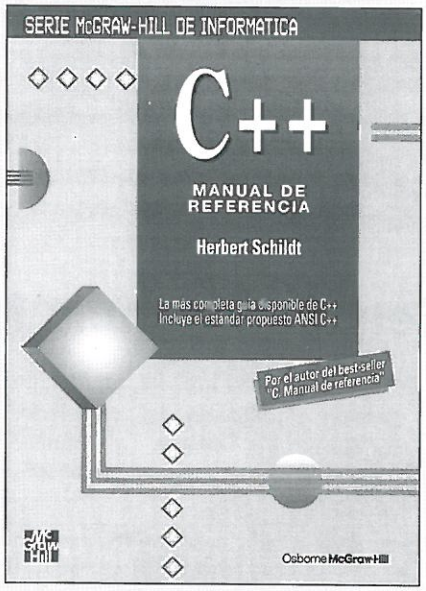
The Desing and Evolution of C++

El diseño y la evolución de C++ es un libro escrito por Bjarne Stroustrup, el propio diseñador del lenguaje C++, en el que se tratan la historia de C++ y el por qué de cada uno de los aspectos de su diseño. La obra se encuentra dividida en dos partes principales: la primera engloba los orígenes y la evolución de C++ a partir del lenguaje C con clases hasta la versión 1.0 de C++, mientras que la segunda cubre las características añadidas al lenguaje desde la versión 1.0. El autor utiliza un estilo claro y bien organizado, a la vez que riguroso, que tiene como resultado un libro de interesante lectura para todos programadores que utilizan habitual u ocasionalmente este lenguaje de programación.

Editorial: Addison Wesley
Autor: Bjarne Stroustrup
471 páginas
Idioma: Inglés
Precio: Consultar

C++ manual de referencia

A pesar del título este libro no se limita a ser un mero manual de referencia del lenguaje C++, si no que podría servir perfectamente para aprender a utilizar uno de los lenguajes de programación más extendidos del momento. Se trata de un texto dirigido a todos los usuarios de C++, desde los principiantes hasta los más avanzados. La primera parte del libro se dedica a la enseñanza del lenguaje C, para entrar de lleno en los detalles específicos de C++ en la segunda parte del mismo. La tercera y última parte está dedicada a un conjunto de ejemplos prácticos de aplicación del lenguaje C++.


SERIE McGRAW-HILL DE INFORMATICA
C++
MANUAL DE REFERENCIA
Herbert Schildt
La más completa guía disponible de C++
Incluye el estándar propuesto ANSI C++
Por el autor del best-seller "C. Manual de referencia"
Osborne McGraw-Hill

Editorial: McGraw-Hill
Autor: Herbert Schildt
610 páginas
Idioma: Castellano
Precio: Consultar

JUGANDO A VIVIR

La educadora animó a Sara, la niña autista, a que diera de comer al oso Pepote. Le mostró cómo llevar la cucharilla a la boca de su osito, una y otra vez. Al cabo de un rato, Sara podía hacer el movimiento por sí misma, pero lo hacía sin ningún entusiasmo. Pronto abandonó. Para ella, el pretendido juego no era más que un ejercicio físico sin significado: llevar mecánicamente una cucharilla entre una taza vacía y la boca de un peluche. No podía trasladar esta iteración sin sentido al plano imaginario: así no se puede dar de comer a un osito. Los estudios sobre el autismo demuestran que los niños autistas carecen de la habilidad cognitiva necesaria para "hacer como si": la capacidad de fingir, lo necesario para representar un papel.

Vivir es representar papeles. Esto es así hasta el punto de que no sabemos cuál es nuestro yo verdadero. Tenemos un papel para hablar con nuestro jefe, otro para seducir a esa persona que nos atrae, incluso otro más para hablarnos a nosotros mismos en el solitario discurso interior de nuestra habitación. Sabemos decir cuáles de nuestras múltiples caretas son más cómodas de llevar, pero siempre nos estará vedado conocer el rostro que hay debajo.

Así que representar múltiples papeles es una de las actividades imprescindibles en nuestra vida de relación. No es extraño, entonces, que los autistas no puedan llevar una vida normal. Pero ¿quién nos enseña a representar papeles? ¿Cómo adquirimos el arte dramático necesario para vivir? Para mí, la respuesta es ésta: jugando. Desde que despierta nuestra autoconsciencia, empezamos a jugar, empezamos a "hacer como si", a convertir una cuchara en un avión y a nosotros en su piloto. Por eso jugar es una cosa muy seria, y fabricar juegos, también.

Un juego es un mundo donde cada uno representa su propio papel, un universo que tiene por leyes físicas las reglas acordadas por los participantes. Como en el mundo, todo lo que hacemos es el resultado de una elección, y todos nuestros actos tienen su consecuencia. Igual que en la vida real, lo que ata nuestros actos a sus efectos sobre el mundo está, en medida variable, zarandeado por el ruido de fondo del azar. Jugando aprendemos que todo acto, incluido el "paso", el acto de no hacer nada, es el fruto de una elección. También descubrimos que la libertad y la responsabilidad son las dos caras de la misma moneda: la conciencia de que, nos guste o no, controlamos nuestro propio destino, y nuestros aciertos y nuestros errores nos pertenecen sólo a nosotros. Punto éste que no deja de ser un concepto de orden moral, y de realidad completamente discutible, pero bueno, a algunos nos gusta jugar a que así es.

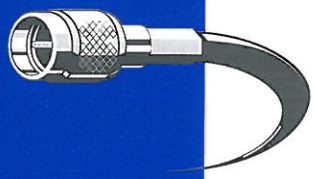
Todo juego es una pequeña vida. Toda vida es un gran juego. La realidad virtual es un cuento, una muleta. Nuestra capacidad innata para meternos dentro de nuestros papeles lleva milenios haciendo todo el trabajo, sin necesidad de cascos o guantes. Sin embargo, los juegos de ordenador están acercando cada vez más las apariencias del juego y de la vida, multiplicando los caminos de huída.

La gran virtud de los juegos, y por supuesto también de los juegos de ordenador, es que podemos salirnos de ellos. Cuando te agobias, pulsas Escape y el pequeño universo del juego se extingue. En la vida sólo hay una forma de salir del juego, y no resulta muy práctica. Otra virtud de muchos juegos es la de dar la capacidad de volver atrás y enmendar los errores. En cambio, la vida no tiene esa opción de menú. Las leyes de la termodinámica dicen que ningún suceso real es reversible. El diario de la vida se escribe con bolígrafo, no con lápiz, y no existe la posibilidad de arrancar páginas. Por otra parte, el tiempo del juego es flexible: el entrenador de baloncesto puede formar una T con sus dos manos y hacer que los relojes se detengan. En la vida, sin embargo, el concepto de "tiempo real" se vive en todo su doloroso significado cuando hay que tomar decisiones rápidas.

La vida es mucho más exigente que los juegos. Por eso los juegos, aunque no hayamos pensado nunca en ello, son siempre, en relación a la vida, aprendizaje o evasión. Y los desarrolladores de los mejores juegos lo saben. Aprendizaje que nos prepara, sin correr riesgos, para el duro juego de opciones que es la vida real. Y evasión que nos permite olvidar, aunque sea por un momento, que la vida no es más que el juego más difícil.

EL LENGUAJE HTML (y III)

Fernando J. Echevarrieta



En el presente artículo se completa el ciclo dedicado al lenguaje HTML con la exposición de las últimas extensiones en fase de borrador o no estándares, como tablas y actualización dinámica de documentos.

En los dos pasados artículos dedicados a HTML se presentaron los fundamentos del lenguaje así las posibilidades de realización de documentos más profesionales a través de imágenes-mapa sensibles y de páginas formulario. En el presente, se introducirán algunas extensiones más artísticas, aunque también necesarias en el mundo profesional de cara a la presentación de una buena imagen publicitaria. En algunos puntos, nunca sin advertirlo, las líneas a seguir se apartarán de los estándares, entre otras cosas porque los mismos estándares se apartan de sí mismos en cada cambio, ya que se encuentran en fase de desarrollo. En definitiva, en el presente artículo se muestra todo aquello que dará un toque de distinción a nuestros documentos.

REALIZACIÓN DE TABLAS: ETIQUETAS

La creación de tablas es uno de los terrenos cenagosos en la definición del HTML. Mucha de la discusión surge de las diferencias entre las dos concepciones que comienzan a distinguirse, europea y americana, de lo que debe ser el el WWW. Para la concepción europea, el sistema debe orientarse principalmente como un medio hacia el contenido, mientras que la europea, más comercial, lo concibe también como un fin. Por ello, el sector americano presiona por incorporar un mayor control de la estética y la presentación a los documentos (en la búsqueda de la publicidad) con lo que trata de imponer estilos físicos en lugar de lógicos (ver discusión sobre generadores WYSIWYG HTML del número anterior). El *W3 Consortium*, por su parte, defiende al HTML como DTD de SGML y, por tanto, lenguaje de definición de la lógica del documento. Sin embargo, para tratar de solventar estos temas, los drafts de HTML3.0 contemplan estilos físicos en las llamadas *hojas de estilo* externas. Sea como fuere, el draft de HTML3.0 ha caducado ya hace tiempo sin convertirse en RFC y el apartado de tablas nacido el 28 de Marzo de 1995 fue revisado nuevamente el 7 de Julio. En este momento y dado el continuo cambio, no se habla ya de draft, sino de *working draft*, cuya última versión a la hora de escribir este artículo es del día 27 de Octubre. Seguramente, cuando vd. lo lea lo habrán cambiado otra media docena de veces. En cualquier caso, siempre podrá acceder a la última versión en el *W3 Documentation Project* en *http://www.w3.org*. Sin embargo, si bien la especificación cambia tan rápidamente, los browsers deben ser programados y distribuídos, con lo que se corre el riesgo de que los estándares lo sean de facto, dictados por un fabricante, en lugar de consenso y nacidos de un comité internacional. Por el momento parece que la voz cantante la sigue teniendo Netscape, ya que incluso la modificación del 7 de Julio, se acercaba más a su forma de implementar las tablas. Sea como fuere, parece ser que la filosofía general en la implementaciópn de tablas esta dictada y será lo que se presentará a continuación con algunos comentarios sobre su posible evolución futura.

Hasta el momento de la aparición del primer draft de HTML3.0, para representar tablas en páginas de WWW se utilizaban tablas en ASCII enmarcadas por la etiqueta *<PRE>...</PRE>* (ver artículo del mes anterior). Desde ese momento, se definen nuevas etiquetas para la representación de tablas delimitadas por la etiqueta *<TABLE>...</TABLE>*. Al igual

que ocurría con las etiquetas dedicadas a forms, las dedicadas a tablas carecen de significado si no aparecen rodeadas por <table>.

La relización de una tabla es más sencilla que la de un form. Así, tras declarar su comienzo, se colocará una etiqueta <tr>...</tr> (*table row*) por cada columna de la misma. En cada una de las filas podrán aparecer celdillas que serán de dos tipos: de cabecera o normales. El contenido de una celdilla normal se enmarca mediante la etiqueta <td>...</td> (*table data*), mientras que el de una de cabecera lo hace mediante la etiqueta <th>...</th> (*table header*). En una celdilla puede aparecer cualquier etiqueta HTML por lo que es posible establecer jerarquías de tablas creando tablas como celdas de otras. Por supuesto, es posible incluir también imágenes.

Por último, se define una etiqueta más denominada <caption>...</caption> que coloca el título a la tabla.

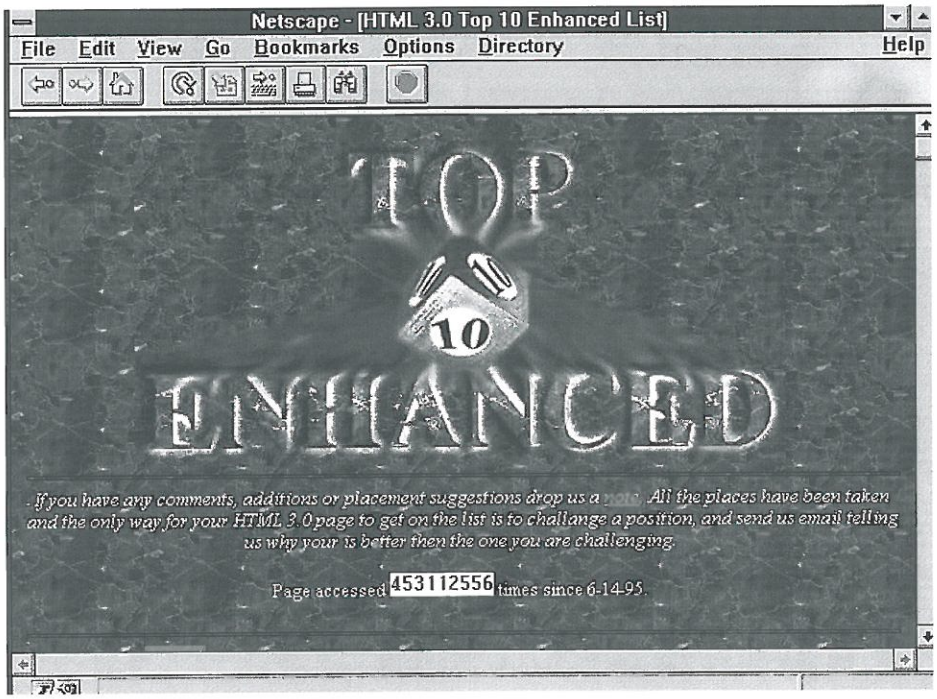
RECUADRO

```
<table [BORDER[=]] [CELLSPACING=]
[CELLPADDING=] [WIDTH=]>...</table>
<tr [align=][valign=]></tr>
<td [align=][valign=][nowrap][colspan=]
[rowspan=]></td>
<th [align=][valign=][nowrap][colspan=]
[rowspan=]></th>
<caption [align=]></caption>
```

Un ejemplo de tabla sencilla, la de la figura 1, sería:

```
<table border>
<tr><th>Columna 1</th><th>Columna
2</th></tr>
<tr><td>Dato 1,1</td><td>Dato
2,1</td></tr>
<tr><td>Dato 1,2</td><td>Dato
2,2</td></tr>
<tr><td>Dato 1,3</td><td>Dato
2,3</td></tr>
</table>
```

donde se puede apreciar se ha empleado el atributo *BORDER* en la etiqueta <TABLE>. Al aparecer, el atributo obliga a que se dibujen los bordes de la tabla que, por defecto no aparecen. En cualquiera de los dos casos, la tabla ocupará lo mismo. El atributo *BORDER* es uno de los que se cuestionan cada día. Así, en NHTML1.1 (Netscape, ver artículo del mes anterior), se emplea de la forma


Netscape - [HTML 3.0 Top 10 Enhanced List]
File Edit View Go Bookmarks Options Directory Help
TOP
10
ENHANCED
If you have any comments, additions or placement suggestions drop us a
All the places have been taken
and the only way for your HTML 3.0 page to get on the list is to challange a position, and send us email telling
us why your is better then the one you are challenging.
Page accessed 453112556 times since 6-14-95.

Ejemplo de página con fondo fotográfico, imagen transparente y colores especificados.

BORDER=<valor> donde valor especifica el grosor del borde de la tabla. En el último draft de HTML3.0, el atributo *BORDER* se emplea para especificar si se deben o no dibujar los bordes y dónde, como indica la tabla adjunta, y la forma se deja para las mencionadas hojas de estilo.

TABLA

Valor	Efecto
none	Suprime bordes
frame	Sólo borde externo
basic	Sólo entre cabecera/cuerpo y cuerpo/pie
rows	Básico y separador de columnas
cols	Básico y separador de filas
all	Todos los bordes

Atributo border en el Draft de HTML 3.0

ATRIBUTOS DE LAS TABLAS

Es en el aspecto de los atributos donde más difieren los drafts de HTML3.0 entre sí y con Netscape. Sin embargo, Netscape está al alcance de todos y es tangible, por lo que, por razones prácticas se hará una exposición de los atributos que gestiona. Además del atributo *BORDER* expuesto, existen otros para ayudar a la definición de la tabla. Así pues, es posible realizar alineamientos del texto en la tabla mediante los atributos:

ALIGN: En la etiqueta <caption> puede tomar los valores *top* o *bottom* e indica si el título se colocará a la cabeza o al pie de tabla respectivamente. En las etiquetas <tr>, <td> y <th> puede tomar los valores *left*, *center* o *right* e indica la forma en que el texto se alineará en las celdas: izquierda, centrado o derecha respectivamente. El draft de HTML 3.0 permite además su uso en la etiqueta <table> para justificar la colocación de las tablas de la misma forma que se hace con las figuras. Sin embargo, Netscape no lo soporta.

VALIGN: Indica la alineación, esta vez vertical del texto en las celdillas y puede tomar los valores *top*, *middle* y *bottom* en las etiquetas <tr>, <td> y <th> para alineamiento superior, centrado e inferior respectivamente.

Por defecto, las alineaciones son a la izquierda en horizontal y centradas en vertical. Un atributo en la etiqueta <tr> cualifica a toda su fila, sin embargo, los atributos que aparezcan en celdas concretas, <td> o <th>, tienen preferancia.

NOWRAP. Se utiliza en las etiquetas <th> y <td> para indicar que las líneas que contienen no deben ser truncadas. Este atributo se encontraba en la especificación del 28 de Marzo pero ha desaparecido en las siguientes aunque Netscape los soporta.

También es posible realizar expansión de celdillas para que ocupen varias filas

y/o columnas en las etiquetas <*th*> y <*td*> mediante los atributos:

COLSPAN: Especifica cuántas columnas debe expandirse la celda.

ROWSPAN: Especifica cuantas filas debe expandirse la celda.

OTROS ATRIBUTOS: CONTROL DE ASPECTO

Para el control físico del aspecto de la tabla, también difieren los atributos empleados por Netscape y los que aparecen en los drafts, que aún presentan algunas inconsistencias con el DTD de SGML. Así pues, centrándonos en Netscape, la etiqueta <*table*> admite:

CELLSPACING: Controla el espacio entre celdas. Netscape toma por defecto el valor 2.

CELLPADDING: Controla el espacio entre el borde y el contenido. Netscape toma por defecto el valor 1.

WIDTH: Trata de ajustar el ancho de la tabla. Los drafts de HTML admiten este atributo en tanto por ciento de la presentación en pantalla. Netscape, además, permite expresarlo en un valor absoluto en pixels.

En las figuras que acompañan al artículo es posible ver el efecto de la utilización de estos atributos. El código necesario para generarlas ha sido inclído en el disco de la revista.

EXTENSIONES DE NETSCAPE

En este momento, Netscape presume de haber incorporado algunas extensiones como propuesta a HTML 2.0 y a HTML 3.0. Respecto a estas últimas, aún no se emplea a fondo lo que hay definido de HTML 3.0 y menos las extensiones propuestas por compañías. Sin embargo, la red se encuentra ya poblada de documentos que hacen uso de las primeras. La mayoría de las extensiones consisten en añadir atributos a las etiquetas ya existentes, aunque también se proponen algunas etiquetas nuevas. En cualquier caso, el uso de estas extensiones no supone una pérdida de compatibilidad ya que, como se explicó hace dos números, cuando una etiqueta o atributo no es identificado, simplemente se ignoran.

RECUADRO

```
<HR [SIZE=] [WIDTH=] [ALIGN=] [NOSHADE]>
<IMG [ALIGN=] [WIDTH=[%]] [HEIGHT=[%]]
[BORDER=] [VSPACE=] [HSPACE=]>
<BR [CLEAR=]>
<CENTER>
<BODY [BACKGROUND=] [BGCOLOR=]
[TEXT=] [LINK=] [VLINK=] [ALINK=]>
<BASEFONT SIZE=>
<FONT SIZE=>
```

Nota: Posibles valores para <IMG ALIGN=>:
left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom.

NUEVOS ATRIBUTOS

Entre las extensiones más simples y utilizadas figuran los atributos del separador horizontal <*HR*> como *SIZE=<valor>*, que determina su grosor; *WIDTH=<valor|tanto_por_ciento>*, que especifica el ancho absoluto o relativo al ancho de la ventana del browser; *ALIGN=<left|right|center>*, que sirve para alinear el separador cuando éste no ocupa todo el ancho; y *NOSHADE*, que le quita la sombra.

Quizá las extensiónes más utilizada por todos los creadores de páginas WWW son los atributos de la etiqueta <*IMG*>, que permiten un extenso control de la situación relativa entre texto e imágenes que tan limitada se encuentra en la versión 2.0 de HTML. Así, se modifican o añaden los siguientes atributos:

ALIGN: Ahora puede tomar nuevos valores como *left* o *right*, que alinean la foto a izquierda o derecha respectivamente permitiendo que todo el texto se distribuya a su lado, no sólo una línea como en la especificación original. El recuadro adjunto refleja otros posibles valores de más dudosa utilidad que se dejan a la experimentación del lector.

VSPACE=<valor> y HSPACE=<valor>: Controlan el margen vertical y horizontal que debe haber entre una imagen y el texto que ahora, con los nuevos valores de *ALIGN*, puede rodearla.

BORDER=<valor>: Permite controlar el grosor del borde que rodea una imagen. Si se indica valor 0, cualquier borde desaparecerá, lo que puede llegar a ser un problema cuando la imagen es un enlace, ya que no será reconocible a simple vista como tal. Sin embargo, puede servir para crear interesantes efectos, como el que se puede observar en las casillas del juego *WEB MINES* en *http://www.dit.upm.es:8000/minas.html* y cuya programación se mostrará en un futuro artículo dedicado a la programación de CGI's. Precisamente, como punto de partida para esta aplicación, se tomará la aplicación cliente-servidor que se explica en el artículo dedicado a programación con sockets en este mismo número de la revista, por lo que, aunque no es necesario, puede ser una lectura recomendable para aquellos lectores que deseen seguir los artículos sobre CGI. En el tablero correspondiente a este juego, cada casilla está constituída por una imagen-enlace sin borde de la forma <*A HREF=...><IMG SRC=... BORDER=0></A*> con el objeto de dar la impresión de que todo el tablero se encuentra integrado en el documento. En las figuras 2 y 3 se puede observar el efecto con borde (como se verá en la mayoría de los browsers) y sin borde de un boceto de tablero.

Columna 1	Columna 2
Dato 1,1	Dato 2,1
Dato 1,2	Dato 2,2
Dato 1,3	Dato 2,3

Tabla con BORDER

Figura 1.

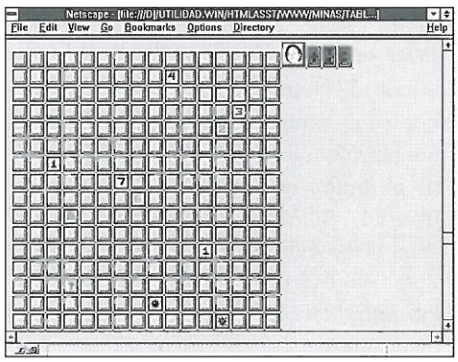


Figura 2. Imágenes-enlace con borde.

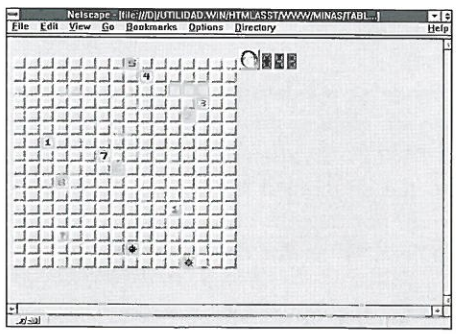


Figura 3. Efecto de BORDER=0 en Netscape.

Por útimo, como complemento a las extensiones anteriores, se amplía la etiqueta *
* que en el estándar produce una ruptura de línea. Ahora, se le añade el atributo *CLEAR=<left|right|all>* que provoca que la ruptura continúe hasta que se encuentre un margen limpio a la izquierda, a la derecha o a ambos lados respectivamente.

UNA NUEVA APARIENCIA PARA EL WWW

Como se puede observar, la mayoría de las aportaciones de Netscape se orientan hacia un mayor control de estilos físicos, es decir, a un control de la forma. Con ellas se pueden lograr páginas de gran sentido estético. Así la tela de araña se encuentra ya poblada de fondos y textos multicolor, distintas fuentes de letras y maquetaciones más o menos complejas.

Entre las nuevas incorporaciones, la etiqueta más usada es *<CENTER>..</CENTER>* que sirve para centrar cualquier elemento que se se sitúe envuelto por ella, ya sea texto o imagen.

El control de los fondos y colores soprende por su sencillez. Simplemente se han añadido nuevos atributos a la etiqueta *<BODY>*:

BACKGROUND=URL: Mediante este atributo se designa por su URL la imagen que servirá de fondo al documento de la misma forma que *SRC=URL* actúa en la etiqueta *<IMG>*. Si la imagen es pequeña se colocará en modo mosaico, como fondo del documento, con lo que se pueden lograr interesantes efectos jugando con texturas.

BGCOLOR=#rrggbb: Si únicamente se desea cambiar el color del fondo sin cargar una imagen, se puede emplear este atributo, cuyo valor es la codificación del color en rgb (red, green, blue) en hexadecimal. Es decir, cada componente puede tomar valores entre 00 y FF. Por poner un ejemplo, el negro sería #000000 (rojo=0,verde=0,azul=0); el blanco, #FFFFFF; el rojo más brillante, #FF0000; y el amarillo más brillante #FFFF00. Es posible generar degradados de color, como fundidos en negro en una película, insertando toda una gama de colores de fondo en el documento. De esta forma, segun los vaya recibiendo el browser irá cambiando el fondo en pantalla. En cualquier caso esta es una práctica no recomendada ya que únicamente debe aparecer una etiqueta *<BODY>...</BODY>* en el documento y no se garantiza su funcionamiento.

TEXT=#rrggbb: No tendría sentido un fondo negro si las letras no pudieran cambiar a otro color. Así, mediante este atributo se controla de forma análoga al del fondo, el color de texto normal (el que no forma parte de un enlace).

LINK, VLINK, ALINK: Modifican el color de los enlaces sin visitar, visitados y el activo respectivamente. Por activo se entiende el que acaba de ser pulsado antes de que aparezca la nueva página. La codificación del color se realiza de la misma forma en rgb: #rrggbb.

Figura 4. Imágenes auto-escaladas.

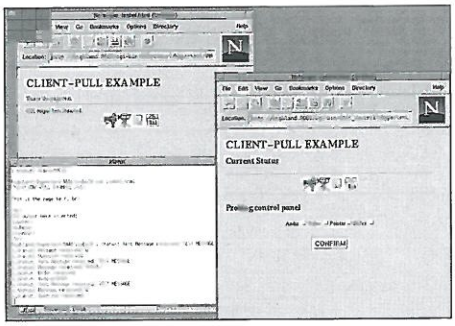


Figura 5. Actualización dinámica de la presentación de un estado.

Un ejemplo de documento podría ser:

```
<HTML>
<HEAD>
<TITLE>COLORES PARA SP</TITLE>
</HEAD>
<BODY BGCOLOR="#000000"
    TEXT  ="#FFFFFF"
    LINK  ="#FFFF00"
    VLINK ="#FF0000"
    ALINK ="#FF00FF">
Este es un documento de prueba con el fondo de color negro, el texto normal en blanco brillante, los enlaces en amarillo brillante y, <a href=highland.dit.upm.es:8000>al pulsarlos</a>, se vuelven malva. Los enlaces ya visitados aparecen en color rojo brillante.
</BODY>
</HTML>
```

Pero no sólo es posible jugar con los colores y la posición de las imágenes. También es posible ajustar el tamaño de las fuentes de letra. Para ello, se añade la etiqueta *<BASEFONT SIZE=número>* que asigna un tamaño a la fuente base que, por defecto es 3.

Columna 1	Columna 2
Dato 1,1	Dato 2,1
Dato 1,2	Dato 2,2
Dato 1,3	Dato 2,3

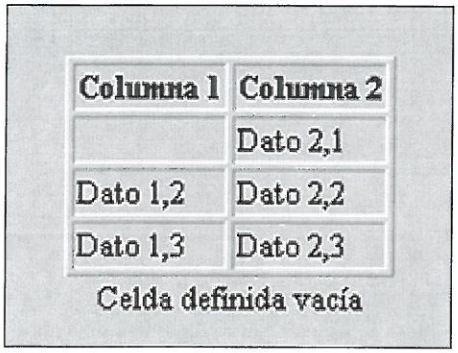
Tabla sin BORDER

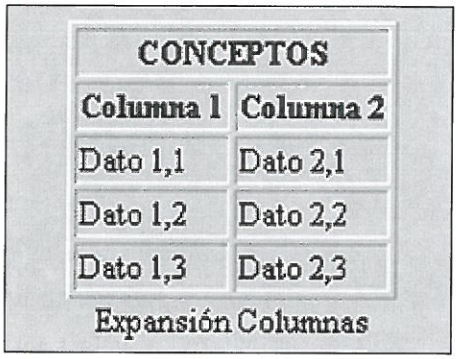
Columna 1	Columna 2
Dato 2,1>	
Dato 1,2	Dato 2,2
Dato 1,3	Dato 2,3

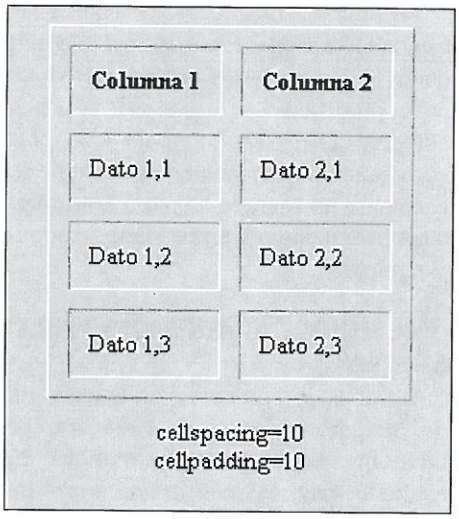
Celda no definida

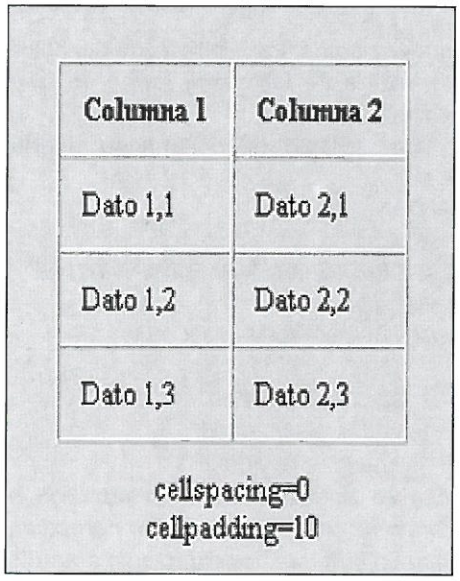
Columna 1	Columna 2
	Dato 2,1
Dato 1,2	Dato 2,2
Dato 1,3	Dato 2,3

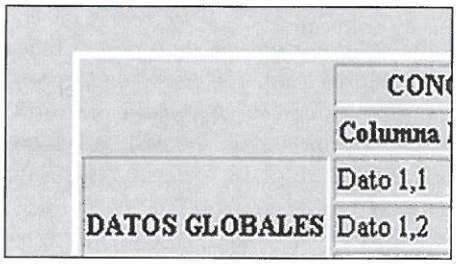
Celda vacía

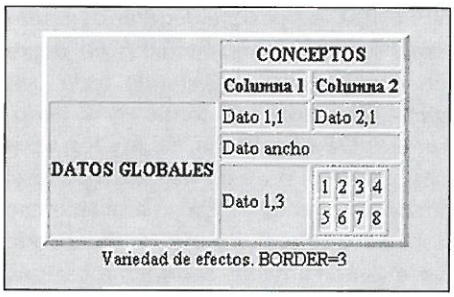


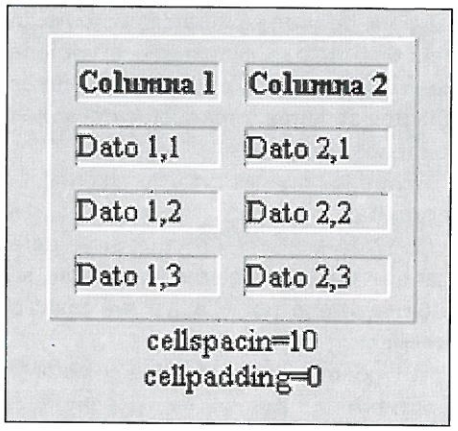












Valores válidos son entre 1 y 7. Para variar el tamaño de una zona determinada del texto, se emplea *<FONT SIZE=número>* donde el número indica la variación relativa de tamaño respecto a la fuente base nuevamente en un rango de 1 a 7.

EFECTOS ESPECIALES CON LAS IMÁGENES

A los mencionados nuevos atributos de la etiqueta *<IMG>* hay que añadir otro dos atributos que indican el ancho y alto respectivamente de una imagen: *WIDTH* y *HEIGTH*. Con ellos se gana en velocidad ya que el browser no necesita cargar la imagen para conocer cuanto espacio debe reservar para ella y, por tanto, muestra más rápidamente el documento. Pero también pueden ser utilizados como un mecanismo para el reescalado de imágenes. Así, el browser mostrará la imagen ajustándose a los parámetros *WIDTH* y *HEIGTH* y no a su tamaño real. De este modo, es posible enviar una imagen pequeña y presentarla como si tuviera un tamaño mayor o, contando con que una imagen grande se encuentra en la cache, presentarla en otras páginas a escala más reducida o, incluso, como un icono. Netscape mantendrá la relación de aspecto si únicamente se indica uno de los dos atributos, pero deformará la imagen si se indican los dos y no la mantienen.

Pero el efecto más interesante surge de la posibilidad de expresar los valores correspondientes a estos atributos como tantos por ciento de la anchura o altura de la ventana del browser. De esta forma, redimensionando la ventana, se logrará un redimensionamiento dinámico de las imágenes, lo que puede ser de gran utilidad para la realización de presentaciones a base de transparencias. Con este mecanismo, las transparencias se ajustarán a la resolución que el usuario tenga en pantalla y este podrá hacer mayor o menor su ventana con objeto de observar una mejor nitidez. Un ejemplo de este efecto puede apreciarse en el servidor del autor en el apartado de fotos de la Third International WWW Conference, Darmstadt, Alemania al que puede accederse desde la página principal: *http://highland.dit.upm.es:8000* y que ejecuta un CGI visor de presentaciones cuya realización también será expuesta en esta revista. En la figura 4 puede observarse una de estas páginas en ventanas de distinto tamaño. Nótese como la única imagen que se reescala es la de la fotografía, y se conserva constante el tamaño de los botones a los que no se ha aplicado los nuevos atributos.

Por último, otra de las propuesta realizadas por Netscape, aunque no ha tenido tanta difusión, es la posibilidad de mostrar imágenes en dos pasadas. Es decir, enviando primero las imágenes a muy baja resolución de manera que presenten una idea global del documento para luego realizar una segunda pasada superponiendo las imágenes a su resolución definitiva. Para ello se define un nuevo atributo: *LOWSRC=URL*, para la etiqueta *<IMG>*, que indica el URL de la imagen de baja resolución que será enviada en la primera pasada. Por ejemplo:

<IMG SRC="Alta_res.gif" LOWSRC="Baja_res.gif">

ACTUALIZACIÓN DINÁMICA

El problema de la actualización dinámica de documentos es uno de los más candentes en el mundo del WWW. Por ello, es aquí donde más discusión existe y donde más soluciones propietarias distintas han aparecido. Gestionado desde el punto de vista del cliente, como una extensión de HTML sin necesidad de programación, destaca por su sencillez la propuesta de Netscape conocida por el nombre de *client pull*. Por ello y por tratarse del browser más extendido, es la que se presenta a continuación.

Mediante la utilización de la etiqueta *<META>* de HTML 3.0, que simula respuestas de HTTP en documentos HTML y debe ser utilizada antes que cualquier otra cosa en el documento, es posible realizar documentos que transcurrido un tiempo en *segundos*, carguen el contenido de un *URL*. La sintaxis es la siguiente:

<META HTTP-EQUIV="Refresh" CONTENT="<segundos>; [URL=<URL>]">

si no se indica *URL*, el efecto será el de volver a cargar el propio documento transcurrido el tiempo indicado.

Esto no tiene sentido si se trata siempre del mismo documento, pero si "algo" hiciera que éste cambiara con el tiempo, se dispondría de un perfecto muestreo de su estado, por ejemplo, la actualización de las cotizaciones de bolsa cada minuto o las entradas y salidas en un ciberforo cada 10 segundos. Con ello, ya se evita que el usuario tenga que pulsar ningún botón para acceder a información actualizada. Otra posible aplicación es la realización de una presentación temporizada en la que el documento A llame al B transcurridos x segundos, éste al C transcurridos y segundos y así sucesivamente, pudiendo convertise en una presentación circular si el último documento llama al primero de nuevo.

Por último, otro de los efectos interesantes es la realización de documentos con audio, en los que no es necesario pulsar sobre ningún enlace para escuchar el sonido. Así, suele ser útil dejar el campo *CONTENT=0*, que provocará el acceso al URL indicado lo antes posible. De este modo, si éste es un fichero de sonido, la presentación del documento y el sonido serán "lo más simultáneas posible".

En *http://highland.dit.upm.es:8000/isabel/gate.html* se puede ver un ejemplo de uso de esta técnica. Para ello, conviene abrir dos ventanas en el browser y con una acceder a la opción *control test page* y con la otra a la de *status test page*. Los botones del panel de control actúan sobre un demonio que guarda un status. La página que muestra el status se actualiza sola cada cierto tiempo reflejando el nuevo status. Para ello, cuando encuentra la etiqueta *<echeva>*, se activa un CGI que pregunta el status al demonio. El aspecto puede verse en la figura 5.

Además de la solución client-pull, Netscape propone otra mucho más eficiente denominada server-push, que es el responsable de la mayoría de las animaciones que pueblan el WWW. Sin embargo, este mecanismo actúa desde el lado del servidor y no forma parte del lenguaje HTML sino de la programación CGI, por lo que se tratará a su debido momento.

	CONCEPTOS	
	Columna 1	Columna 2
DATOS GLOBALES	Dato 1,1	Dato 2,1
	Dato 1,2	Dato 2,2
	Dato 1,3	Dato 2,3

Expansión filas y columnas. BORDER=3

BIBLIOGRAFÍA

- An Exploration of Dynamic Documents
http://home.mcom.com/assist/net_sites/pushpull.html
- Extensions to HTML
http://home/mcom.com/assist/net_sites/html_extensions.html
- HTML3.0 vs NHTML1.1
http://www.iaf.nl/%7Eabigail/HTML/Myth/toc.html
- Introduction to HTML 3.0
http://www.w3.org/hypertext/WWW/MarkUp/html3/intro.html
- The HTML3 table model. W3C working draft 27 Oct 1995
http://www.w3.org/pub/WWW/TR/WD-tables-951027.html
- Tables
http://www.w3.org/hypertext/WWW/MarkUp/html3/tables.html
- Guia de Diseño de documentos HTML. Marcas de Tabla
http://borabora.univalle.edu.co/disenno/html_design.html

CONCLUSIÓN

Con este artículo de la serie de WWW se cierra la etapa dedicada a HTML aunque no está aún todo dicho. En la versión 2.0 de Netscape en su fase de BETA 3 a la hora de escribir este artículo se incluye un nuevo elemento denominado *frame*, que de tener éxito llenará de subventanas las ventanas de nuestros browsers. También se incorpora la posibilidad de realizar *client-side imagemaps* que coinciden exactamente con la especificación que se mencionaba el mes pasado de *Spyglass Mosaic*, con lo que se les puede augurar un interesante futuro. El W3 Consortium sigue trabajando en su especificación de HTML 3.0, así que en otro momento habrá que volver sobre el HTML para presentar además de estas nuevas incorporaciones, sus capacidades matemáticas y, si por fin se estandariza, la nueva versión 3.0 y la definición de sus hojas de estilo. Por el momento, el mes que viene comenzaremos una nueva etapa explorando aspectos como instalación de servidores y programación CGI.

CONTACTAR CON EL AUTOR

Para cualquier clase de duda, aclaración, comentario, sugerencia o crítica, se anima al lector a que se ponga en contacto con el autor a través de carta a la redacción de la revista o, preferiblemente, mediante:

E-Mail Internet: echeva@dit.upm.es
E-Mail Compuserve: 100646,2456
WWW: http://highland.dit.upm.es:8000
ftp: highland.dit.upm.es

ALGORITMOS RECURSIVOS

José C. Remiro

La recursividad es una técnica que permite expresar definiciones que se autocontienen. aunque es relativamente fácil de comprender e implementar, presenta algunas características que la hacen ineficaz si no se utiliza de forma adecuada.

La mayoría de los lenguajes de programación permiten a un procedimiento o función hacer referencia a sí mismos dentro de su definición, a esta característica se la denomina recursividad o recursividad. Incluso, es posible que un procedimiento haga referencia a otro y éste último haga referencia al primero, así se dirá en el primer caso que el procedimiento es directamente recursivo y en el segundo que se trata de procedimientos con recursividad indirecta. La relación en este último caso será más difícil de determinar.

El efecto que se provoca con la recursividad directa es la llamada de un procedimiento al mismo procedimiento, a su vez esta llamada provoca una nueva llamada y así sucesivamente. Si el diseño del procedimiento ha sido correcto y los parámetros se adecuan a la situación, llegará un momento en que una llamada al procedimiento no provoque una nueva llamada, en esta caso la llamada acaba su ejecución pasando el control a la llamada anterior, que de nuevo, terminará devolviendo el control a la llamada anterior, y así sucesivamente, hasta que se llegue a la llamada inicial, que terminará.

La recursividad permite, al igual que las instrucciones iterativas, la repetición de una serie de cálculos que guardan cierta similitud. Aunque es cierto que las instrucciones iterativas están más cercanas a la estructura de las computadoras que la recursividad, ésta es muy útil para la realización de procedimientos y funciones que pueden definirse de forma recursiva o bien en la construcción de procedimientos y funciones que tengan que tratar con estructuras previamente definidas como recursivas (por ejemplo, los tipos de datos listas y árboles).

Un primer ejemplo

En el cuadro 1, se pueden observar las funciones *factorial_rec* y ***factorial_iter***, ambas calculan dado un número entero positivo su factorial, mientras que la primera función realiza el cálculo de forma recursiva, la segunda realiza el cálculo de forma iterativa. La función *factorial_rec* se basa en la definición: "El factorial de 0 es 1. El factorial de un número superior a 0 es dicho número por el factorial de su predecesor". La función *factorial_iter* se basa en la definición: "El factorial de un número es el producto de los números comprendidos entre 1 y dicho número, ambos inclusive".

Este ejemplo básico, permite observar que un procedimiento iterativo tienen su equivalente en un procedimiento recursivo y a la inversa. Además, la fun-

```
funcion factorial_rec (n: entero): entero

 inicio

  Si n = 0
    entonces
      factorial_rec = 1
    en otro caso
      factorial_rec = factorial_rec (n-1) * n

 fin


funcion factorial_iter (n: entero):entero
i, acum: entero

inicio
acum:= 1
Desde i = 1 hasta n
      acum = acum * i
factorial_iter = acum
fin
```

Figura 1.

ción *factorial_iter* descompone el problema utilizando el producto de los números comprendidos entre 1 y el número del que se quiere calcular su factorial, acumulando dicho resultado en una variable intermedia. La función *factorial_rec*, por el contrario, está implementada de forma que supuesto conocido el factorial de n-1, calcular el factorial de n es inmediato. Es importante darse cuenta que el tipo de datos sobre el que se aplica la función *factorial_rec* es el mismo para todas las llamadas y que la llamada que se provoca en su cuerpo se aplica sobre un dato más sencillo (su predecesor), de tal forma que las sucesivas llamadas se aplicarán a datos cada vez más cercanos a 0, por lo que es evidente que la cadena de llamadas terminará cuando se produzca la llamada a *factorial_rec(0)*.

La recursividad es posible gracias a las reglas que gobiernan la validez de los identificadores. Así, cuando se llama a un procedimiento se crean un conjunto de variables, constantes y definiciones locales al procedimiento que no son accesibles desde fuera del procedimiento, de tal forma que cuando se llama a un procedimiento recursivamente se crea un conjunto de nuevos valores para las variables, pero teniendo una localidad diferente, coexistiendo con los valores creados anteriormente. Lo importante, es que cada procedimiento manipulará sus correspondientes valores y no los valores de las anteriores llamadas al procedimiento. Esta idea se encuentra expresada en el cuadro 2, donde se puede observar las diferentes llamadas cuando se calcula el factorial de 4 mediante la función *factorial_rec*. La operación representada por

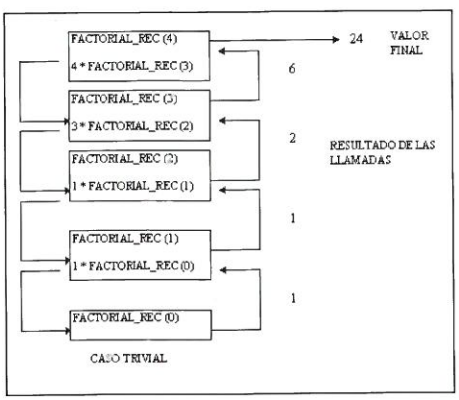


Figura 2.

*n*Factorial_rec(n-1)* queda pendiente hasta que se obtiene el valor de *Factorial_rec(n-1)*, evidentemente, ésta no devolverá su valor hasta que se llegue a obtener el valor de *Factorial_rec(0)* que es el caso trivial (caso que no genera ninguna otra llamada). Una vez obtenido, las llamadas precedentes, junto con sus operaciones, podrán realizarse.

Diseño recursivo

El desarrollo del anterior ejemplo muestra varias ideas acerca de como obtener un procedimiento recursivo. A la hora de realizar el diseño de este tipo de procedimientos se han de tener en cuenta algunas consideraciones. En primer lugar, los parámetros de estos procedimientos deben ser del mismo tipo, pero de tal forma que la llamada recursiva tenga parámetros más "simples" en cuanto que se acercan al caso o los casos triviales .

En cuanto al cuerpo del procedimiento, dispondrá de una o varias instrucciones condicionales en las que se distinguirán los casos triviales (asociados a las instrucciones que no generarán una nueva llamada al procedimiento) de los que no lo son. En la parte asociada a los casos no triviales, habrá un conjunto de instrucciones que prepararán los datos para proceder a una nueva llamada al procedimiento. Finalmente, tras la llamada al procedimiento, se procederá a operar con los datos obtenidos con el fin de obtener un tratamiento para los datos que acompañaban a la primera llamada.

Por lo visto hasta ahora, parece ser que se dispone de un diseño muy apropiado para ser aplicado a las situaciones en que se disponga de definiciones

Definición de la función de Fibonacci

Fibonacci (0) = 0
Fibonacci (1) = 1
Fibonacci (n+1) = Fibonacci (n) + Fibonacci(n-1), para n > 1

```
funcion Fib_rec (n: entero): entero

inicio
 Si n=0
  entonces
   Fib_rec = 0
  en otro caso
   Si n = 1
     entonces
      Fib_rec = 1
     en otro caso
       Fib_rec = Fib_rec (n-1) + Fib_rec (n-2)
fin
```

Figura 3.

o estructuras recursivas. Si bien esto es cierto, hay situaciones en las que no es muy adecuado utilizar este diseño. En los cuadros 3 y a se encuentra el seudocódigo correspondiente a la función de Fibonnaci implementada de forma recursiva *(Fib_rec)* y a su implementación de forma iterativa *(Fib_iter)*. Analizando el número de llamadas para calcular *Fib_rec(n)*, vemos que se duplican cada vez que **n** crece en una unidad, por lo que se puede decir que el número de llamadas crece de forma exponencial a medida que crece **n**, además hay llamadas duplicadas, este último hecho se evita en la función ***Fib_iter***.

Es evidente, por tanto, que cuando se encuentre un algoritmo iterativo sencillo se preferirá utilizar éste. Por tanto, incluso el ejemplo de la función ***factorial_rec*** aun siendo un buen ejemplo para explicar la recursividad, es un mal ejemplo en cuanto a eficiencia en comparación con su homólogo iterativo. Entonces, ¿cuándo utilizar algoritmos recursivos?. En primer lugar, decir que si un algoritmo puede implementarse de forma recursiva, se puede utilizar una pila y transformar éste en iterativo, aunque el precio a pagar es el de un algoritmo más difícil de comprender. De forma que, un algoritmo recursivo puede ayudar a encontrar su correspondiente algoritmo iterativo. Por otro lado, cuando las llamadas generadas por el procedimiento recursivo sea suficientemente pequeño está justificado su uso, así como para el tratamiento de estructuras que puedan definirse de forma recursiva.

De recursivo a iterativo

Como se ha indicado anteriormente,

```
funcion Fib_iter (n: entero): entero
 i, siguiente, anterior: entero

 inicio
 i = 1
 siguiente = 1
 anterior = 0
 mientras i < n
  siguiente = siguiente + anterior
  anterior = siguiente - anterior
  i = i + 1
 fin mientras
 factorial_rec = factorial_rec (n-1) * n
 Fib_iter = siguiente
 fin
```

Figura 4.

```
procedimiento ret_quick_sort(var a: array (N) real)
 procedimiento orden(izq, dcha: entero)
  sup, inf :entero
  aux, inter: real
 Inicio
  inf= izq
  sup= dcha
  aux= a[(izq+dcha) div 2]
 Repetir
  Mientras a[inf] < aux
     inf= inf + 1
  Fin mientras
  Mientras a[sup] > aux
     sup= sup - 1
  Fin mientras
  Si inf <= sup
    inter= a[inf]
    a[inf]= a[sup]
    a[sup]= inter
    inf= inf + 1
    sup= sup - 1
  Fin si
 hasta inf > sup
 Si izq < sup
  orden(izq, sup)
 Fin si
 Si inf < dcha
  orden(inf, dcha)
 Fin si
Fin
Inicio /*del proc. ORDEN*/
 orden(1, n)
Fin
```

Figura 5.

es posible diseñar un algoritmo de forma recursiva y posteriormente trasladarlo a un diseño iterativo.

En el cuadro 5 se encuentra el algoritmo recursivo de ordenación *quick sort* para la ordenación de un array. Este algoritmo se basa en el siguiente principio: "Si se dispone de dos conjuntos de elementos de tal forma que todos los elementos del primer conjunto son menores que cada uno de los elementos del segundo elemento, se pueden obtener ordenados todos los elementos, ordenando cada uno de los conjuntos por separado".

Es claro que la definición de este algoritmo es recursiva, pues la ordenación del array se hace ordenando dos particiones mas pequeñas que tengan la anterior propiedad, a su vez estas particiones se pueden subdividir y ordenar por separado, siguiendo este proceso hasta obtener una partición de un solo elemento, en cuyo caso está ordenada (caso trivial de la recursividad). De esta manera, si se conocen los límites de las particiones y éstas son adecuadas la ordenación resulta muy sencilla.

También, hay que señalar la estructura que presenta este algoritmo. Como se indicó anteriormente, en primer lugar hay una preparación para las llamadas al procedimiento, ésta se produce en la instrucción *repetir* más externa, en ella se distribuyen los elementos en dos zonas respecto a un elemento elegido, en este caso el elemento central de la secuencia a ordenar. Una vez preparado el terreno para la recursividad, se comprueba si las particiones obtenidas contienen más de un elemento, en cuyo caso se producirá la llamada. Esta fase del algoritmo está representada por las instrucciones *Si*. El caso trivial de recursividad es, por tanto, aquel en que la partición disponga de tan solo un elemento, es decir, no hay nada que ordenar.

En el cuadro 6 se encuentra el algoritmo *quick sort* pero en versión iterativa. La idea central es la misma, dividir el array en particiones y ordenarlas independientemente. Pero en este caso no es posible proceder con la siguiente iteración producida por una nueva partición sin guardar los límites de la otra partición que se produce. Por tanto, se deben guardar los límites de la partición que no se va a tratar de manera inmediata para proceder posteriormente a su tratamiento. Sin embargo, el orden de recuperación de los límites de las particiones que quedan pendientes por ordenar no pueden ser recuperadas en cualquier orden, de tal forma que hay que proceder a recuperarlas en el orden inverso en que fueron generadas, es decir, las últimas que fueron generadas hay que tratarlas en primer lugar. Para realizar la gestión de los límites de las particiones pendientes se dispone de un array para simular una estructura de pila. La variable *cima* indicará el mayor elemento ocupado del array que se utiliza para simular la pila. Para apilar los límites de una nueva partición se añadirá uno a la variable *cima* y en dicha posición del array se introducirán los límites de la partición a tratar posteriormente. Para desapilar, bastará con recuperar los valores del elemento del array señalado por la variable *cima* y decrementar en uno el valor de la cima. Para poner en marcha este algoritmo, se apila la primera partición pendiente, esto es, la totalidad del array (la secuencia de instrucciones que se encuentran fuera de la instrucción *repetir* más externa). La instrucción *repetir* más externa está dedicada a desapilar la siguiente partición que quedó pendiente y a generar a partir de ésta dos nuevas particiones más pequeñas, tratando una de estas particiones y apilando la otra partición que quedará pendiente.

El algoritmo *quick sort* iterativo puede ser mejorado. Si se analiza el mejor de los casos, en el que se obtienen como particiones las mitades de la anterior partición, el número de itera-

```
procedimiento quick_sort_iter (var a: array (n) real);
 inf, sup, iz, de: entero
 aux, inter: real
 cima : entero
 elemento: registro
             lim_iz, lim_de: entero
 pila: array [1..n] elemento
inicio
cima = 1
pila[cima].lim_iz = 1
pila[cima].lim_de = 100

Repetir
    iz = pila[cima].lim_iz
    de = pila[cima].lim_de
    cima = cima - 1
    Repetir
      inf = iz
      sup = de
      aux = a[(iz+de) / 2]
      Repetir
        Mientras a[inf] < aux
           inf = inf + 1
        Mientras a[sup] > aux
           sup = sup - 1
        Si inf <= sup
          entonces
             inter = a[inf]
             a[inf] = a[sup]
             a[sup] = inter
        Fin Si
      Hasta inf > sup
      Si inf < de
         entonces
           cima = cima + 1
           pila[cima].lim_iz = inf
           pila[cima].lim_de = de
      Fin Si
      de = sup
    Hasta iz >= de
Hasta cima = 0
fin
```

Figura 6.

ciones para ejecutar el algoritmo está cercano al logaritmo de n, donde n es el número de elementos a ordenar. Por el contrario, si una de las particiones que se genera cada vez que se realizan particiones es de longitud 1, entonces la eficiencia se reduce, pues harían falta n elementos en el array que simula la pila para albergar la información de estas particiones. Este hecho, puede evitarse seleccionando la partición más pequeña para ser tratada a continuación, almacenando en la pila los límites de la partición más larga, evitando así el crecimiento desmesurado de las ordenaciones que quedan pendientes. En el cuadro 7 se encuentra el algoritmo quick sort iterativo mejorado, en éste ya no se selecciona la partición izquierda para ser tratada a continuación, si no que se comprueba la longitud de las particiones que se han establecido, seleccionando la más pequeña para ser tratada a continuación y los límites de la partición más larga se almacenan en la pila para un tratamiento posterior.

El estudio de este ejemplo pone de manifiesto la importancia de reducir las llamadas de tipo recursivo, pues el sistema cuando ejecuta un procedimiento recursivo utilizará de manera implícita una pila para guardar las variables locales de las llamadas pendientes, de tal forma que si no se es cuidadoso a la hora de realizar el diseño recursivo puede ocurrir que se desborde la pila impidiendo la correcta ejecución del procedimiento.

```
procedimiento quick_sort_mej (var a: array (n) real);
 inf, sup, iz, de: entero
 aux, inter: real
 cima : entero
 elemento: registro
             lim_iz, lim_de: entero
 pila: array [1..n] elemento
inicio
cima = 1
pila[cima].lim_iz = 1
pila[cima].lim_de = 100

Repetir
    iz = pila[cima].lim_iz
    de = pila[cima].lim_de
    cima = cima - 1
    Repetir
      inf = iz
      sup = de
      aux = a[(iz+de) / 2]

      Repetir
        Mientras a[inf] < aux
            inf = inf + 1
        Mientras a[sup] > aux
            sup = sup - 1

        Si inf <= sup
          entonces
              inter = a[inf]
              a[inf] = a[sup]
              a[sup] = inter
        Fin Si
      Hasta inf > sup
      Si sup - iz < de - inf /*mejora del quick sort*/
          entonces
           Si inf < de
            entonces
              cima = cima + 1
              pila[cima].lim_iz = inf
              pila[cima].lim_de = de
            Fin Si
            de = sup
          en otro caso
            Si iz < sup
              entonces
                cima = cima + 1
                pila[cima].lim_iz = iz
                pila[cima].lim_de = sup
            Fin Si
            iz = inf
      Fin Si
    Hasta iz >= de
Hasta cima = 0
fin
```

Figura 7.

Backtracking

Hay un gran número de problemas en los que no se puede aplicar una solución analítica, es decir, la solución no puede ser expresada de acuerdo a una serie de cálculos más o menos complejos.

Por el contrario, es necesario aplicar métodos de tanteo de forma sistemática. En este tipo de problemas generales se suele descomponer el proceso en tareas parciales, estos constituyen un paso hacia la solución buscada. Tras la ejecución de un paso son posible dos opciones: que éste forme parte de la solución buscada, por lo que será posible progresar hacia la obtención de la solución, o bien, que este paso no forme parte de la solución, por lo que habrá que proceder a deshacer el paso dado seleccionando otro paso diferente al seleccionado con anterioridad.

Para diseñar algoritmos que utilicen la anterior técnica es muy útil la recursividad, pues lo que se intenta es ordenar cada uno de los pasos hacia la solución, cuando un paso no es el adecuado, se aplicará el mismo proceso pero con el siguiente paso dentro del orden establecido. Será en este punto donde se aplicará la recursividad. Como ya se comentó con anterioridad, la recursividad es adecuada cuando el número de llamadas es suficientemente pequeño, por lo que habrá que realizar un esfuerzo extra para intentar reducir el número de llamadas recursivas. Así, seleccionar adecuadamente la estructura de datos para representar el problema y los parámetros de los procedimientos recursivos, será una de las tareas más importantes a realizar en estos algoritmos.

Se presenta como ejemplo de backtracking el problema de las ocho reinas. Consiste en disponer ocho reinas en un tablero de ajedrez de tal forma que éstas no se amenacen entre sí, recordemos que una reina amenaza a las piezas situadas en una casilla que esté en la misma fila, columna o diagonal que la casilla en la que se encuentra. Puesto que en una misma columna no puede haber mas de una reina, se utilizará un array de ocho posiciones, cada una de los valores del índice representará el número de columna donde se dispondrá cada una de las damas (debe haber exactamente una dama por columna), de tal forma que además de representar la columna donde se encuentra la dama también representa a la propia pieza. Cada una de las posiciones de dicho array, contendrá el número de fila donde se colocará la reina. Así, si este array contiene en la posición 5 el valor 3 indicará que en la fila 3 columna 5 se encuentra una dama.

La recursividad permite incluir llamadas de un procedimiento en su propio cuerpo

Por último, habrá que seleccionar una representación para el tablero. Parece ser que lo más adecuado es un

```
procedimiento inicia_reinas
  indice: entero
  correcto: logico
  columnas: array [ 8] entero
  filas: array [8] logico
  diagonal_sup : array [15] logico
  diagonal_inf : array [15] logico

procedimiento poner_reina (i: entero;
                           var c: logico)
    prueba: entero
  inicio
    prueba = 0
    Repetir
      prueba = prueba + 1
      c = falso
      Si filas[prueba] y diagonal_sup [i+prueba -1] y
         diagonal[i-prueba + 8]
        entonces /*realizar la prueba*/
          columnas[i] = falso
          diagonal_sup[i+prueba-1] = falso
          diagonal_inf[i-prueba+8] = falso
          Si i < 8
            entonces
              poner_reina(i+1, c)
                                  Si no c
                                  /*prueba no válida. Deshacer el cambio*/
                                    entonces
                                    columnas[i] = verdadero
                                    diagonal_sup[i+prueba-1] = verdadero
                                    diagonal_inf[i-prueba+8] = verdadero
                                  Fin Si
                                 En otro caso
                                   c = verdadero
                                Fin Si
                            Fin Si
                          Hasta c o (prueba = 8)
                          Fin /*del procedimiento poner_reina*/

                        inicio
                        /*inicializar las variables para comenzar*/
                         Desde indice = 1 hasta 8
                           filas [indice] = verdadero
                         Desde indice = 1 hasta 15
                           diagonal_sup [indice] = verdadero
                           diagonal_inf [indice] = verdadero
                         Fin Desde
                         poner_reina(1, correcto)
                         Si correcto
                            Desde indice = 1 hasta 8
                             Escribir ('Una reina en posición ',
                                       'fila ',columnas [indice],
                                       'columna ', indice)
                        fin
```

Figura 8.

array bidimensional. Esta representación conduciría a tener que inspeccionar los 64 escaques de que dispone el tablero para comprobar, tras disponer una dama en una posición, si ésta amenaza o no a otra dama que se dispuso con anterioridad, por lo que la comprobación tras realizar una posible disposición seria muy complicada. En vez de utilizar esta representación se dispondrá de tres arrays, el primero, de ocho posiciones, para representar si la fila en la que actualmente se va a disponer la reina se encuentra ya ocupada, el segundo y tercero , de 15 posiciones, para comprobar si el escaque donde se va a situar la reina se encuentra ya amenazado diagonalmente. Los tres arrays tendrán el valor **verdadero** si no se encuentran amenazados por las reinas que se han dispuesto con anterioridad y **falso** en caso contrario. Por último, solamente hace falta hacer corresponder a cada par (**fila** , **columna**) el número de diagonal que amenaza, pero es fácil darse cuenta que las cantidades **fila** + **columna** y **fila** - **columna** son constantes en todos los escaques que pertenecen a la misma diagonal.

Una vez seleccionada la representación de la información a utilizar en el algoritmo, el modo de proceder será el siguiente. Se situará la primera reina en el primer lugar disponible, en principio en la primera fila pues no está amenazada por ninguna otra reina (sencillamente no hay todavía ninguna otra situada). Para buscar acomodo a la reina de la segunda columna, habrá que encontrar un escaque que no esté amenazado por la anterior. Esto se cumple si los contenidos correspondientes a los arrays de la fila y las diagonales tienen el valor **verdadero**. Tras situar esta reina (modificando los valores correspondientes a la fila y diagonales amenazadas) se intentará disponer a las siguientes de igual forma. Llegará un momento en que las reinas colocadas amenacen todos los escaques de la fila correspondiente a una nueva reina, por lo que habrá que deshacer el paso anterior, es decir, volver a situar en un lugar adecuado a la reina que se situó en último lugar. El procedimiento ***poner_reina*** del cuadro 8 terminará entonces con la variable **fila** igual a ocho, devolverá el control al procedimiento que efectuó la llamada. Éste volverá a situar la dama que se situó en último lugar en el siguiente lugar disponible. Una vez encontrado éste, se volverá a llamar al procedimiento ***poner_reina***, que intentará de nuevo situar la reina que no pudo colocarse anteriormente.

Es posible traducir un procedimiento recursivo a iterativo y viceversa

Es importante darse cuenta que cuando no es posible situar una reina no sólo se provoca la búsqueda de una nueva posición para la dama que se situó anteriormente, pues ésta a su vez intentará colocar en las siguientes filas la reina correspondiente, pudiendo ocurrir de nuevo que esto no sea posible, lo que provocará un nuevo desplazamiento de la reina que se situó con anterioridad a ésta y así sucesivamente.

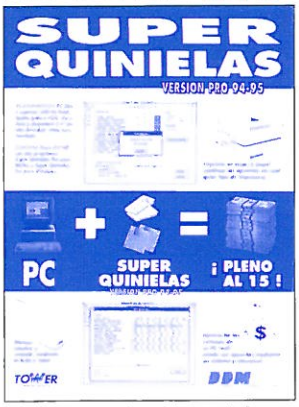
Otro aspecto que puede ayudar a la comprensión del algoritmo, es recordar lo señalado al principio de este artículo: cada llamada recursiva a un procedimiento mantiene ocultos los valores de sus variables locales al resto de los procedimientos (la variable ***fila*** es local al procedimiento recursivo ***poner_reina***). Así, cuando falla un intento de colocación de una reina, se pueden deshacer los cambios en los arrays que marcan las filas y las diagonales como amenazadas, liberándolas y avanzando a la siguiente posición que podría ocupar la reina, gracias a que la variable ***fila*** contiene la última posición donde se intentó situar ésta. El parámetro correcto, indicará si la reina que se intentó colocar a continuación de la actual a través de la llamada recursiva al procedimiento fue satisfactoria o no. Si no fue satisfactoria se procederá a buscar una nueva posición para la reina correspondiente.

Por último, indicar como este algoritmo intenta localizar las posiciones de las reinas proponiendo situaciones para posteriormente deshacerlas si no son las adecuadas para llegar a la solución buscada, siendo este hecho lo que caracteriza a la técnica del backtracking. Además, se muestra una forma de como simplificar el conjunto de pruebas a realizar para obtener la solución, pues al comenzar el algoritmo se elimina todo intento de disponer a las reinas en columnas iguales, por lo que se reduce el número de llamadas recursivas y por tanto se mejora en mucho la eficiencia del algoritmo, siendo éste un criterio necesario a la hora de implementar algoritmos recursivos.

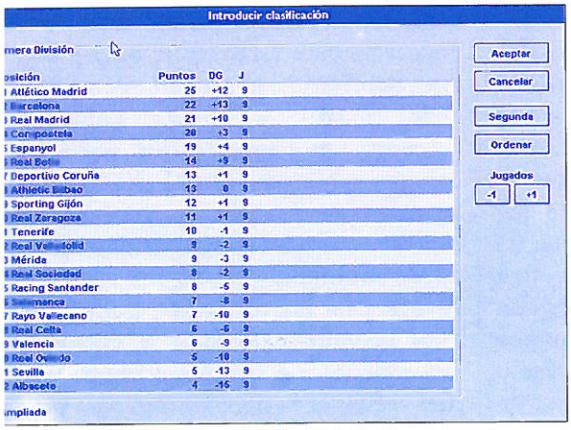
UPER QUINIELAS 3.0

TRES TEMPORADAS FABRICANDO MILLONARIOS

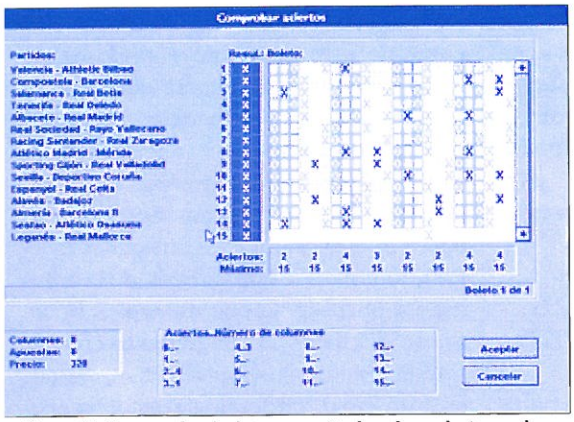
)3 el comienzo de Super nielas.



1994 segunda versión del programa.

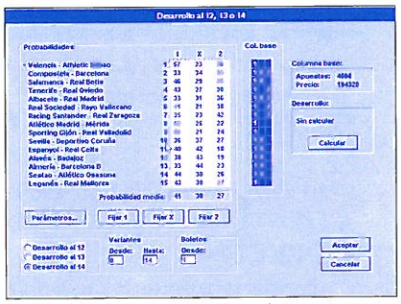


a la liga de primera y segunda división jornada a jornada, las opciones de Super Quinielas.

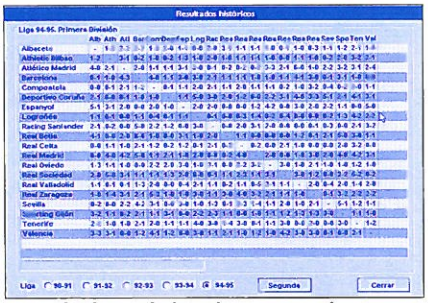


ntifique fácilmente los boletos premiados de cada jornada.

PROGRAMA PARA PC
2.495 ptas.
SUPER QUINIELAS
VERSION 3.0
LIGA 95-96
PC 386 o superior, 640 Kb RAM, tarjeta VGA, disco duro, disquetera 3.5" HD, ratón recomendado.
Disco 3.5" HD con dos programas: Super Quinielas Pro para MsDOS y Super Quinielas Pro para Windows.
BANCO DE ESPAÑA
DIEZ MIL
X2
¡NO HACE IMITACIONES!
DDM



Posibilidad de desarrollos al 12,13 ó14.



Base de datos de las últimas cinco ligas.

es sistemas de cálculo diferentes: histórico, últimos resultados, por clasificación actual.
educción al 12, 13 y 14.
ase de datos de consulta de los últimos 5 años: primera y gunda división.

- *Compruebe sus boletos premiados de la manera más rápida.*
- *Ideal para las peñas quinielísticas.*
- *Permite apostar mediante disquete.*
- *Válido para los próximos años / opción de nueva liga.*
- *Dos versiones por el precio de una: DOS y WINDOWS*

CON LA GARANTÍA DE DDM *DIGIT AL DREAMS MULTIMEDIA*

licita SUPER QUINIELAS 3.0 enviando este cupón o llamando al teléfono (91) 741.26.62 de 9 a 14 y de 15:30 a 18:30, o por Fax: (91) 320 60 72

ieo que me envíen: ☐ **SUPER QUINIELAS VERSIÓN 3.0 por 2495 ptas. + 250 ptas. de gastos de envío.**

ombre y apellidos.....Domicilio.....Población.....

ovincia.....C.P .....F. de nacimiento.....Profesión.....

FORMA DE PAGO:

Talón a ABETO EDITORIAL ☐ Contra-reembolso Teléfono ..... Firma ,

] Giro Postal nº.....de fecha.....

] Tarjeta de crédito VISA nº ☐☐☐ ☐☐☐ ☐☐☐ ☐☐☐

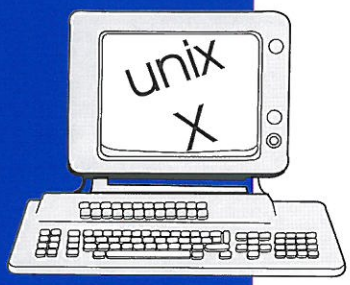
cha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

ABETO

Rellena este cupón y envíalo a:
ABETO EDITORIAL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.

ARQUITECTURAS CLIENTE-SERVIDOR

Fernando J. Echevarrieta



El mes pasado se abrió la primera puerta de las comunicaciones UNIX mediante la interfaz socket. Éste, un repaso de la misma y la programación de un juego en red servirán también como escusa para repasar algunos conceptos sobre procesos y programación en shell.

La interfaz socket no sólo es la primera puerta que se abre en el mundo de las comunicaciones UNIX. Su presencia en multitud de sistemas heterogéneos es una de las claves de la portabilidad de programas y, apoyada sobre una torre de protocolos común dota a un sistema de carácter de *abierto*. A través de este mecanismo, será posible ejecutar el cliente que se va a presentar como ejemplo en un PC con Linux y el servidor en una SPARC con SunOs, dos máquinas con arquitecturas y fabricantes dispares que ni siquiera mantienen el mismo convenio de representación interna de la información. Y esto, por poner sólo un ejemplo.

El mes pasado se presentó un gran número de conceptos relacionados con comunicaciones, protocolos, API's de comunicaciones y casi todas las llamadas al sistema relacionadas con la interfaz socket. Si no han quedado aún asentados todos los conceptos, este mes se revisitarán mediante un ejemplo práctico.

REPASO DE ARQUITECTURAS CLIENTE-SERVIDOR MEDIANTE SOCKETS

El número anterior exponía cómo la comunicación entre máquinas puede realizarse en función de dos principios: orientada a conexión o mediante datagramas. En el primer caso, se establecía una conexión entre los interlocutores que quedaba fijada hasta su liberación, mientras que en el segundo, el traspaso de información se realizaba a través de mensajes individuales desde un origen hasta un destino. Conceptualmente no cabe más división, aunque desde el punto de vista de la implementación mediante el IPC de Berkeley se pueden distinguir tres *modus operandi*:

1. Servidores orientados a conexión: como representa la figura 1, el servidor crea un socket (*socket*), le asigna una dirección (*bind*), lo prepara para escuchar (*listen*) y acepta llamadas (*accept*). Las llamadas que aparecen con una 'b' en una estrella son bloqueantes, es decir, dejan el proceso a la espera. Así, tras el *accept* el proceso queda dormido. La llamada *accept* devuelve un nuevo socket, por lo que todo servidor orientado a conexión requerirá, al menos, dos sockets, uno para recibir conexiones y otro para procesarlas. Cuando un cliente desea comunicarse, crea su socket (*socket*), y establece una conexión (*connect*) al puerto establecido. Es únicamente a partir de este momento cuando existe la conexión y durará hasta que se libere (*close*). Este es el modelo que se utilizará en el ejemplo.
2. Servidores de datagramas conectados en el cliente: como representa la figura 2, el servidor crea su socket (*socket*), lo enlaza a una dirección (*bind*) y espera la recepción de mensajes (*recvfrom*). El cliente, análogamente crea su socket y llama a *connect*. Sin embargo, en este caso, en lugar de establecerse una conexión, el único efecto es que el software de protocolos interno apunta la dirección de destino para que todos los mensajes enviados mediante *write* lleguen a su destino.
3. Servidores de datagramas no conectados en el cliente: son prácticamente idénticos a los anteriores. La única diferencia es que al no realizar el cliente la llamada *connect*, los mensajes deben ser enviados mediante *sendto* indicando destino, como indica la figura 3.

SERVIDORES ITERATIVOS Y CONCURRENTES

A partir de este momento, se pueden distinguir dos tipos de servidores: iterativos y concurrentes.

Un *servidor iterativo* leerá por el el socket devuelto por *accept*, procesará la llamada, si es necesario escribirá y mantendrá un diálogo por él, lo cerrará y volverá a llamar a *accept* para recibir otras peticiones. De esta forma, en cada iteración atenderá a un único cliente. El servidor ejemplo de este artículo será un servidor iterativo.

Un *servidor concurrente*, por el contrario, constará de un *maestro* que será el que realice un *accept* y al recibir una petición generará un servidor *esclavo* mediante un *fork*. Será el servidor esclavo el que procesará la petición, cerrará el socket y terminará. El maestro, una vez arrancado el esclavo cerrará su copia del socket y realizará un nuevo *accept*. En un determinado momento, podrá haber un maestro con distintos esclavos atendiendo concurrentemente a diferentes clientes. Por tanto, se tratará de *servidores multiproceso con concurrencia real*. Existe otra modalidad de servidores concurrentes monoproceso en los que la concurrencia es aparente y de los que se hablará en un futuro artículo.

UN EJEMPLO PRÁCTICO

Con objeto de asentar conceptos, se estudiará como ejemplo una aplicación cliente-servidor basada en el popular juego del buscaminas del que se supone todo el mundo conoce las reglas. A través de la misma se verá cómo las cosas no son tan complejas como parece al exponer ciertos conceptos. El programa no tiene otras pretensiones, por lo que, como se verá, el interfaz deja bastante que desear y no se han protegido contra errores las llamadas al sistema ni el protocolo empleado. Los listados a los que se hará referencia se encuentran en el disquete de la revista agrupados y comprimidos en el fichero: *spminas.tgz*. Para extraerlos, bastará teclear

tar xvzf spminas.tgz

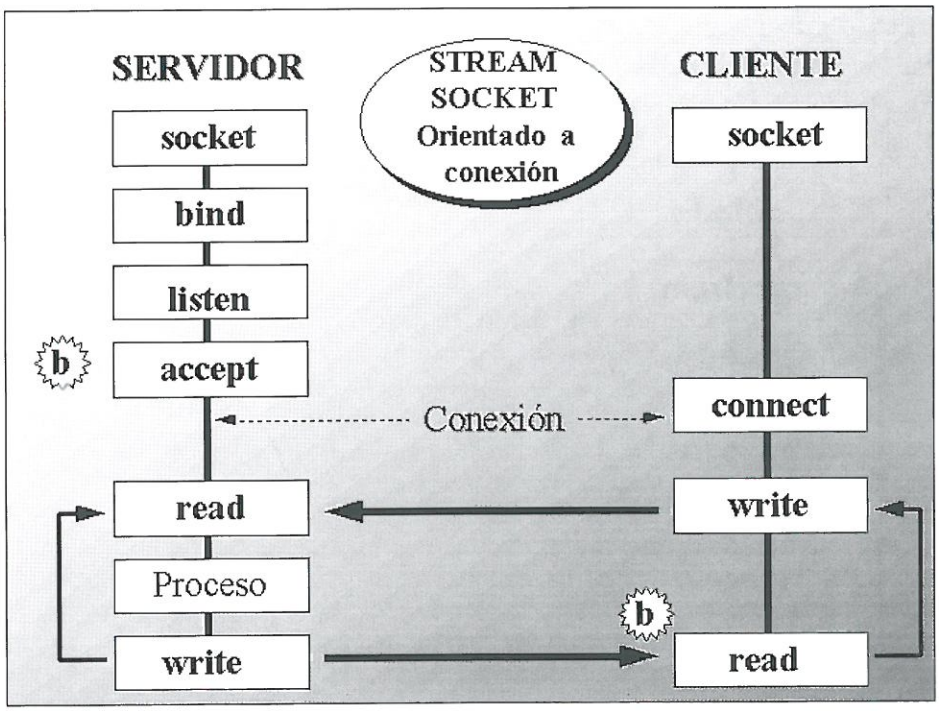
si se dispone del *tar* de GNU o, si no,

gunzip spminas.tgz
tar xvf spminas.tar

Para el desarrollo de la aplicación se ha partido de una versión sencilla en modo texto cuyas fuentes están en *minas.c*. La estructura del programa, muy simple, es autoexplicativa. Se trata de una conversión de un código C para DOS en el que primeramente aparecen las definiciones de tres funciones necesarias para su traspaso a UNIX (generadores aleatorios y borrado de pantalla). El programa consiste simplemente en un bucle en el que se realizan las acciones:

1. Preguntar las coordenadas de la casilla a pisar (entre 1 y 10)
2. Pisar la casilla
3. Mostrar resultado

El programa, sólo cuenta con tres funciones: *minar()*, que genera el tablero y reparte aleatoriamente las minas; *pisar()*, función recursiva que cambia el estado de pisada o no pisada de la casilla elegida y, en caso de que no aporte información, explora las de su alrededor; y *mostrar()*, que muestra en pantalla las actualizaciones del tablero de pisadas. En el disquete se ha incluido un fichero *makefile* que genera los ejecutables para este programa así como para la aplicación cliente-servidor. En caso de que se realice una compilación con el flag *MONITORIZAR*, la función *mostrar()* también mostrará otro tablero con la posición de las minas, lo que será bastante interesante en el lado del servidor.



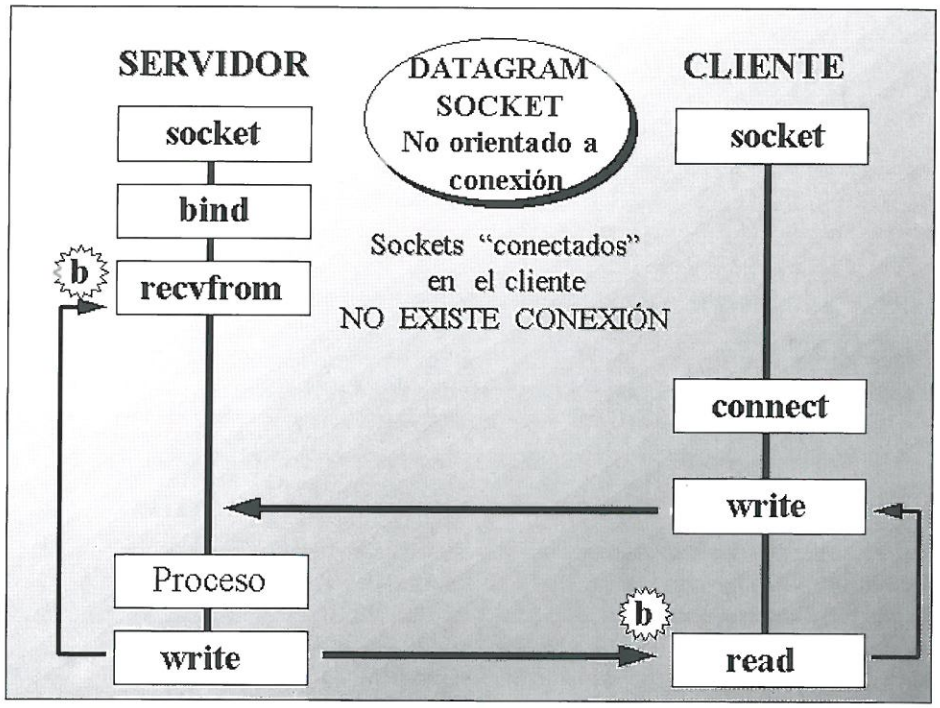
Servicio orientado a conexión

LISTADO 1

```
Escucha()
{
        mineserver.sin_family=AF_INET;
        mineserver.sin_addr.s_addr=INADDR_ANY;

        /* Crea el socket*/
        s=socket(AF_INET, SOCK_STREAM, 0);
        if (s<0)
        {
                perror("server:socket");
                exit(1);
        }

    /* Lo engancha al puerto */
        if (bind(s, (struct sockaddr*)&mineserver,
sizeof(mineserver)) < 0)
                {
                perror("server:bind");
                exit(1);
        }

                /* Y escucha */
        listen(s,1);

    /* Comunica su numero de puerto */
        c=sizeof(mineserver);
        getsockname(s,(struct sockaddr *)
&mineserver,&c);
          printf("mineserver: Escuchando en el puerto
%d\n",
                ntohs(mineserver.sin_port));
}
```

DIVISIÓN DE UNA APLICACIÓN

A la hora de diseñar una aplicación cliente-servidor, una de las primeras tareas a realizar es decidir qué partes formarán parte de la parte servidora y cuáles lo harán de la parte cliente. Esta labor, aunque a primera vista pueda no presentarse como algo muy importan-



Servicio de datagramas "conectado" en cliente

te, es un factor determinante en el funcionamiento de la aplicación. Para ello, habrá que considerar diversos factores. Algunos de ellos constituyen normas elementales del diseño de aplicaciones distribuidas, y se estudiarán el próximo mes, en el que se explicará el mecanismo de *RPC*'s de SUN. Sin embargo, una parte importante de estos factores depende de cada aplicación específica y, quizá en mayor medida, del entorno en que vaya a residir o el comportamiento que se espera de ella.

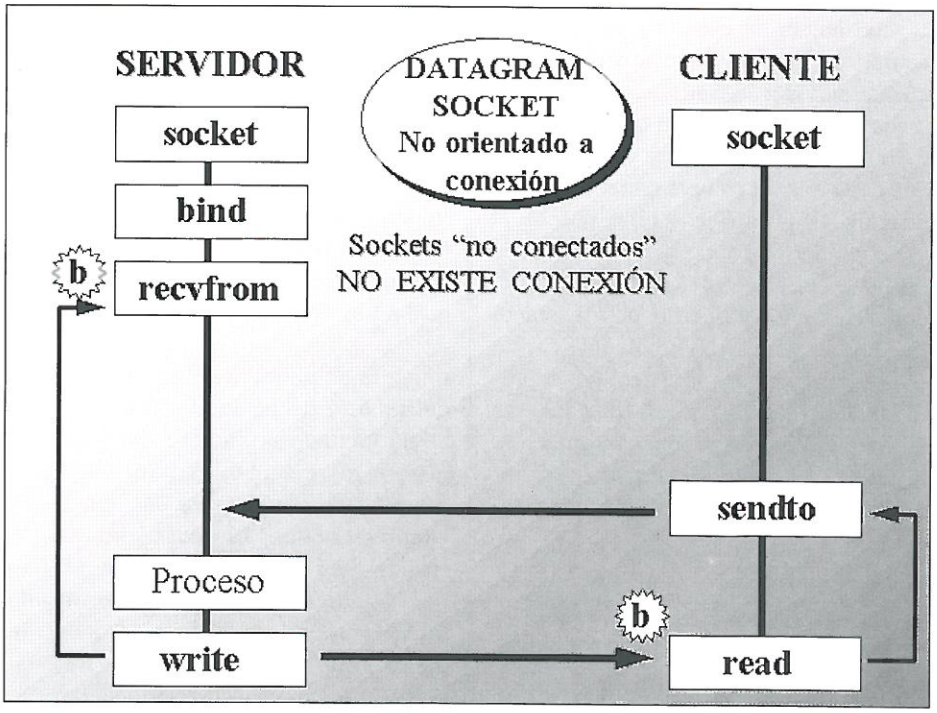
La división que se va a realizar este mes, se va a ver influida en gran medida por un objetivo final: el de su posterior integración en un servicio de WWW. Así pues, sin perder su sentido en sí misma, servirá como punto de partida en los capítulos correspondientes a CGI de la serie dedicada a WWW en esta misma revista. Si bien las labores encargadas a cliente y servidor, así como el protocolo de comunicaciones diferirían si se persiguiera optimizar otro objetivo, la forma de implementar las partes y el protocolo serán idénticas.

ANÁLISIS ORIENTADO A LA COMUNICACIÓN

El diseño de una aplicación distribuida puede realizarse atendiendo a dos metodologías: orientación a la *aplicación* u orientación a la *comunicación*. Sobre estos aspectos se profundizará el próximo mes, cuando se realice una división orientada a la aplicación. De momento, nos centraremos en la metodología orientada a la comunicación que comienza por el paso previo de la definición del *protocolo* que comunicará el cliente con el servidor y continúa realizando sobre él la construcción de la aplicación. Un protocolo no es sino un conjunto de reglas que rigen una comunicación. Cualquier cliente podrá solicitar información al servidor siempre que cumpla estas reglas, siempre que se ajuste al protocolo. En otro caso, se podrá realizar una conexión al puerto de servidor, pero al no conocer el protocolo, el intercambio de datos carecerá de significado, del mismo modo que llamar a un teléfono de Alemania es inútil si no se habla alemán.

Un protocolo serio queda definido por dos elementos: el *servicio* que proporciona y las *PDU*'s (*Protocol Data Unit*) que genera (en castellano *UDP*'s). La mayor parte de las veces, se implementa como un *autómata*, por lo que su comportamiento se representa mediante un *diagrama de estados*. A través de estos diagramas se representan los estados en que pueden encontrarse los interlocutores así como los estímulos que provocan una transición de estado, generalmente, la recepción mensajes del protocolo (aunque pueden ser otros, como un *time-out*).

Calificar de protocolo y aplicar estos conceptos al mecanismo de comunicación que aquí se va a definir es más bien pretencioso. Sin embargo, se ha presentado esta terminología con fines didácticos y con el último objetivo de destacar la importancia del análisis previo por encima de la generación de



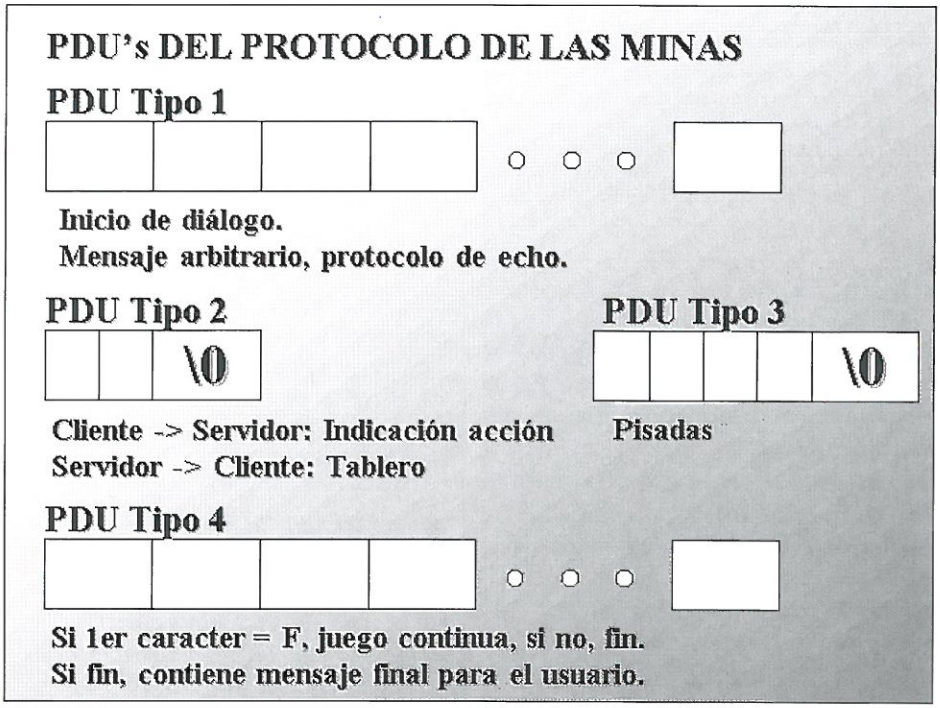
Servicio de datagramas no "conectado" en cliente.

código a la hora de programar comunicaciones.

DEFINICIÓN DEL PROTOCOLO DE LAS MINAS

Para comenzar con el ejemplo se ha definido un "diagrama de estados" del juego del buscaminas que se puede apreciar en la figura 4. No se trata con rigor del diagrama del que se hablaba en el párrafo anterior ya que mezcla estados y mensajes y no determina el comportamiento de cliente y servidor, pero sí presenta cual será el funcionamiento de la aplicación. Así, en una primera fase "COMIENZO DEL DIALOGO" se establecerá y "sincronizará" la comunicación. Tras ella, el cliente podrá realizar una entre tres acciones: transmisión de las coordenadas de una pisada, petición del estado del tablero en un determinado momento y orden de finalización al servidor. El servidor, por su parte, responderá con el estado del tablero completo (tras pisar una casilla, si procede) o, simplemente, muriendo.

Como se puede apreciar, la división realizada, no es del todo lógica ni óptima desde el punto de vista de la comunicación. En una primera aproximación, lo lógico sería que el cliente almacenara el tablero de minas y el servidor el de pisadas. De esta forma, la comunicación desde el servidor hasta cliente sólo llevaría una casilla por pisada en lugar de todo el tablero, lo cual supone un uso más racional de la red. Sin embargo, como se mencionaba anteriormente, este programa será integrado como aplicación CGI en un servidor de WWW y, en ese entorno, la aplicación cliente no podrá almacenar ningún estado y sólo podrá realizar una consulta cada vez que se lance.


PDU's DEL PROTOCOLO DE LAS MINAS
PDU Tipo 1
Inicio de diálogo.
Mensaje arbitrario, protocolo de echo.
PDU Tipo 2
\0
PDU Tipo 3
\0
Cliente -> Servidor: Indicación acción
Pisadas
Servidor -> Cliente: Tablero
PDU Tipo 4
Si 1er caracter = F, juego continua, si no, fin.
Si fin, contiene mensaje final para el usuario.

"PDU's" del protocolo de las minas.

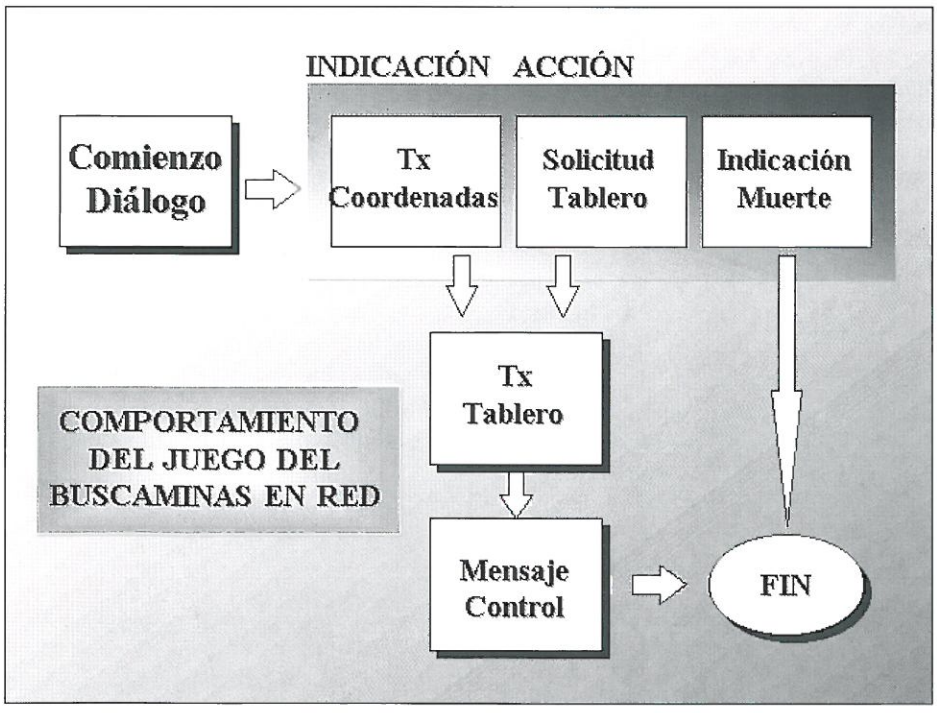


Diagrama del comportamiento de la aplicación en red.

Así pues, atendiendo a razones específicas de un caso concreto, se ha decidido que todo el estado de la partida será almacenado en el servidor. El cliente se limitará a realizar una consulta, recibir el estado global, mostrarlo en pantalla y morir. Para una nueva consulta, habrá que lanzar otro cliente con las nuevas coordenadas. Con objeto de eliminar este tedio, se facilita en el disco un programa en shell *shminas* que se encarga de preguntar al jugador las coordenadas y lanzar los clientes con las preguntas.

Una vez definida la división de la aplicación y el comportamiento del sistema, será necesario especificar los mensajes que transmitirá el protocolo. Es decir, las PDU's. Éstas se han representado en la figura 5.

En ella se pueden distinguir cuatro conjuntos de PDU's:

1. PDU de tipo 1. Longitud: indefinida. Transporta un mensaje arbitrario. Es decir, el cliente enviará cualquier cosa al servidor y esperará que regrese lo mismo.

2. PDU de tipo 2. Longitud: 3 octetos. En el sentido cliente-servidor indica el valor de una coordenada expresada con dos dígitos (entre 1 y 10) y terminada en el carácter nulo \0. Se admiten dos coordenadas fuera de rango con un significado especial:

LISTADO 2

```
LeeSock(int *x, int *y)
{
    /* Acepta conexiones*/
      minelen=sizeof(mineserver);
      ns=accept(s,INADDR_ANY,&minelen);
      if (ns<0)
            {
          perror("server:accept");
          exit(1);
      }

    /* Recibe un mensaje del cliente y lo reenvia*/
      c=read(ns,buf,sizeof(buf));
      write(ns,buf,c);

            /* Recibe las coordenadas */
              read(ns,buf,3);
              *x=atoi(buf);

              read(ns,buf,3);
              *y=atoi(buf);
}
```

x=0,y=0: Solicitud de tablero sin realizar pisada.
x=1,y=0: Señal de terminación.

3. PDU de tipo 3. Longitud: 4 octetos. Indica el número de casillas de las que se conoce su estado (pisadas) terminada con el carácter nulo \0.
4. PDU de tipo 4. Longitud: indefinida. Transporta un mensaje de estado. Si el primer octeto es una letra 'F' indica que el juego continúa. En caso contrario, ha finalizado.

Como se puede apreciar, la especificación del protocolo aquí realizada es del todo informal, imprecisa y ambigua. No obstante, debe entenderse más como una explicación del funcionamiento del programa que como una especificación que, por otra parte no tiene sentido formalizar mucho más en un mecanismo tan simple. Sin embargo, la definición de lenguajes de especificación formal de protocolos es unos de los campos de investigación en telemática en la actualidad.

EXPLICACIÓN DEL CLIENTE

El cliente utilizará un socket de dominio internet (*AF_INET*) con el objeto de poder realizar una comunicación a través de red. Para ello, se declara una estructura *sockaddr_in*, cuya estructura puede observarse en la figura 6, a la que se llama *mineserver* y se rellenan sus campos. El puerto es conocido por ser parámetro del *main()* ya que el cliente responde a la sintaxis: *client <x> <y> <puerto>*.

Obsérvese que la dirección IP asignada e esta estructura, *127.0.0.1*, es una dirección ficticia que corresponde siempre a la máquina local. Por el momento, se emplearán direcciones IP "cableadas" en las fuentes del programa, aunque el lector siempre podrá utilizar un parámetro de programa para referirse a una u otra. Para el empleo de nombres de máquinas habrá que esperar al artículo dedicado a bases de datos de red. Obsérvese también que se ha hecho uso de la rutina de conversión de enteros a formato de red *htons* que se explicó en el número anterior:

```
struct sockaddr_in mineserver;
(...)
mineserver.sin_family=AF_INET;
mineserver.sin_port =htons(port);
mineserver.sin_addr.s_addr=inet_addr("
127.0.0.1");
```

El proceso comienza mediante la creación de un socket en la que se ha especificado que se desea un servicio fiable orientado a conexión (*SOCK_STREAM*). Debido a que el nivel de transporte de Internet únicamente dispone de un protocolo con estas características (*TCP*), se coloca un 0 en el número de protocolo:

```
s=socket(AF_INET,SOCK_STREAM,0);
```

Tras ello, y siguiendo la estructura que muestra la figura 1, se solicita conexión mediante *connect*. Obsérvese como se realiza un casting de la estructura *sockaddr_in* a *sockaddr*. Se podría tratar directamente con una estructura de este último tipo, aunque el buen estilo indica hacerlo de este modo:

```
connect(s,(struct sockaddr *)&mineser-
ver,sizeof(mineserver)
```

El tercer paso es notificar al servidor que se está dispuesto a recibir los datos. El servidor contestará reenviando el mensaje de saludo. En el ejemplo, las reglas de comienzo corresponden a un protocolo de *echo* ("te contesto lo mismo que me envías). Obsérvese que esta parte de sincronización podría eliminarse ya que no se comprueba lo recibido. En general, no se ha realizado ningún tipo de verificación ni protección de la comunicación (confiando en el TCP) ni de las llamadas al sistema, pero se ha incluido para aportar ideas sobre etapas comunes en una comunicación.

```
write(s,"ENVIA EL TABLERO",sizeof-
("ENVIA EL TABLERO"));
read(s,buf,sizeof("ENVIA EL TABLE-
RO"));
```

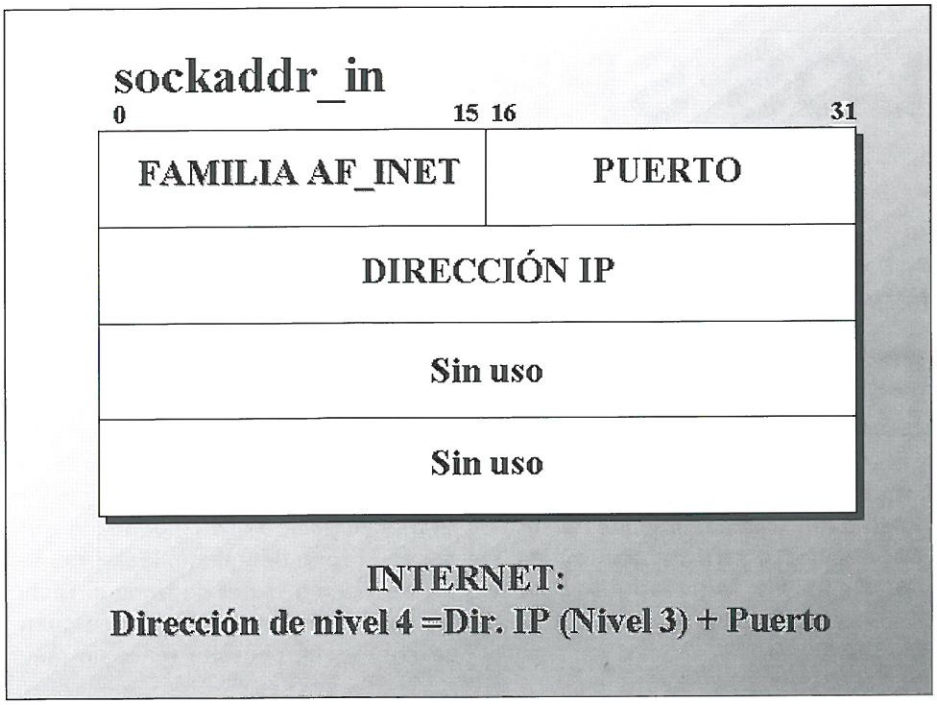
Tras ello, se envían las coordenadas codificando una pisada, solicitud de estado o señal de exterminio como se indicó anteriormente:

```
sprintf(buf,"%2d",x);
write(s,buf,3);

sprintf(buf,"%2d",y);
write(s,buf,3);

if ((x==1) && (y==0))
{
  printf("Enviada senal de exterminio al
servidor\n");
  exit(0);
}
```

Para la comunicación se ha empleado *write* y *read*. Las primitivas se usan del mismo modo que se haría en accesos a ficheros. Y análogamente, al tratarse de un *STREAM SOCKET*, se trata con un *stream* o flujo de comunicación, por lo que habrá que cuidarse de no leer en el buffer más de lo necesario, en este caso, exactamente la cantidad enviada. Cada PDU sólo tiene significado para los protocolos que las intercambian. Para los protocolos de nivel inferior, no son más que datos a transportar y no son analizadas. Así, TCP transmite un flujo de bits entre un socket y otro y no distingue los límites de las PDU's del protocolo de las minas que está en un nivel superior.

A partir de aquí, se transfiere el tablero. Al tratarse de un *stream socket*, se podría leer todo el stream y después procesarlo para extraer de él la información. En el caso del ejemplo, aprovechando que el formato es fijo, se lee en cada paso de un bucle exactamente los bytes correspondientes a cada casilla del tablero evitando un análisis posterior.

```
for (x=0; x<=Xmax; ++x)
  for (y=0; y<=Ymax; ++y)
  {
     read(s,buf,3);
          tablero[x][y]=atoi(buf);
  }
read(s,buf,4);
pisadas=atoi(buf);
```



Estructura sockaddr_in.

Tras ello, se recibe el mensaje de control, esta vez, de longitud indefinida. Aunque para el protocolo únicamente tenga contenido semántico el primer carácter, que será el que indique si el juego continúa o ha terminado, este mensaje no es arbitrario ya que será el que se muestre al usuario en este ultimo caso:

```
read(s,buf,sizeof(buf));
if (buf[0] == 'F') mostrar(0); else mostrar(1);
```

Por último, como si de un fichero se tratara, se cierra el socket:

```
close(s);
```

EXPLICACIÓN DEL SERVIDOR

En el arranque del código del servidor se puede apreciar como éste se pasa automáticamente a background si no se ha compilado con la opción *MONITORIZAR*. Para ello, hace uso de la primitiva *fork()* (ver número 15 de la revista). En este caso, como todo buen servidor, pasa a convertirse en un *demonio*, desligado del terminal. Para proporcionar un mayor control al usuario y por si éste tuviera que acabar con el demonio mediante el comando *kill*, el *PID* del hijo (el que queda como demonio), aparece en pantalla. Es el proceso padre, ligado al terminal, el que escribe todo antes de morir. En otro caso, es bastante probable que la shell mostrara su prompt antes de que el hijo terminara de escribir, por lo que datos y prompt podrían mezclarse:

```
if ( (pid=fork()) != 0 ) {
   printf("mineserver: PID del demonio: %d\n",pid);
  printf("mineserver: OK\n");
  exit(0);
}
```

Tras el arranque y tras *minar()* el tablero, el servidor activa el socket de *Escucha()*, procedimiento que se puede observar en el listado 1, siguiendo las pautas que indicaba la figura 1. Si tiene alguna duda, acuda al artículo del mes pasado o contacte con el autor. Donde en la aplicación inicial había entrada de datos mediante *scanf*, ahora hay un *LeeSock()* (listado 2) y donde antes había *mostrar()* para mostrar el estado en pantalla, ahora hay *MuestraSock()* para mostrar el estado a través del socket. Si en la aplicación inicial se hubiera definido un módulo de biblioteca con los procedimientos de E/S estándar y ahora se hubiera definido otro con los de E/S mediante sockets y con el mismo nombre, bastaría realizar una compilación condicional con uno u otro módulo para tener la aplicación original o el servidor actual. De esta forma, podría incluso haberse conservado el objeto de la parte relativa al juego. En este caso, se ha conservado el procedimiento original *mostrar()* para la opción de monitorización. Así, se puede asistir desde el servidor a la partida que está ejecutando un jugador desde otro lugar.

EL INTERFAZ DE SHELL

El interfaz de shell para el cliente es autoexplicativo. Si tiene dudas, consulte el número 14 de la revista. En cualquier caso, este fragmento admite dos comentarios:

```
if client $x $y $1 > /dev/null
then
  until client $x $y $1 | tee /dev/tty | grep -q HAS
  do
  (...)
  done
fi
```

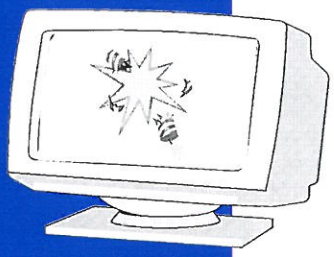
Si la primera vez que se ejecuta *client* (sentencia *if*), falla (por ejemplo, el servidor no está lanzado), el programa termina. No produce pisada si hay éxito ya que x=0 e y=0. El bucle *until* se ejecuta hasta que *grep* acierta. Esto se produce cuando la salida de *client* contiene una de las cadenas "HAS GANADO" o "HAS MUERTO". La opción *-q* se emplea para que *grep* no escriba la cadena encontrada y el comando *tee* para que la salida de client además de *grep* se dirija al terminal.

EJECUCIÓN DE LA APLICACIÓN

Para la generación de la aplicación bastará teclear *make all*. El servidor se lanzará simplemente tecleando *mineserver*. Al ejecutarse, presentará en pantalla el número de puerto por el que atiende. Ejecute el programa *shminas* con ese número como parámetro y ¡a jugar!

CONCLUSIÓN

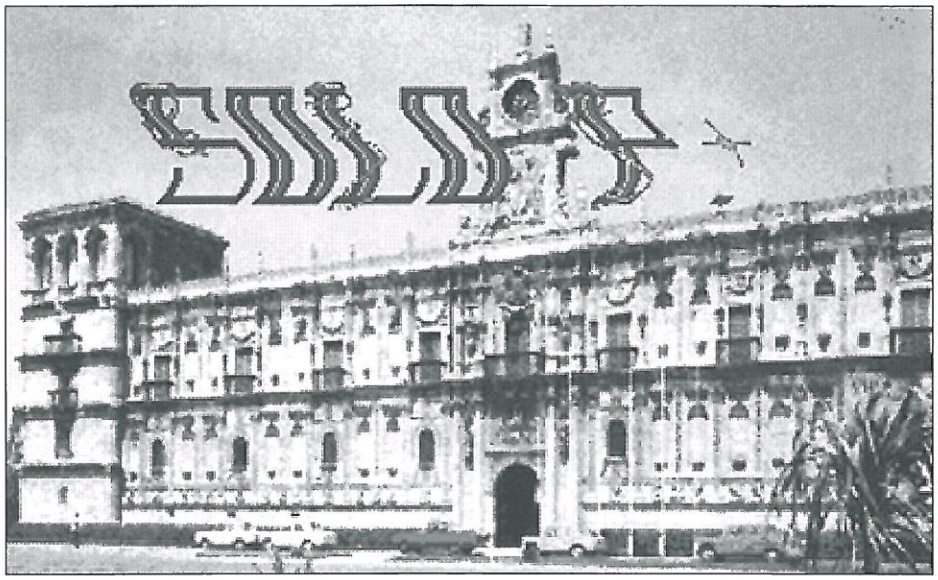
Con este artículo y el del mes anterior, se ha abierto la primera puerta a las comunicaciones desde el mundo UNIX. Dados los primeros pasos a través de la interfaz socket, en los próximos dos números se subirá un peldaño para presentar la programación mediante RPC's. Sin embargo, el principal problema no radica en el manejo de las llamadas al sistema sino en la metodología de diseño de comunicaciones y aplicaciones distribuidas. Sería interesante conocer las opiniones de los lectores al respecto de si profundizar más o no en estos temas, preferiblemente mediante carta a la redacción de la revista.



LOS SPRITES

Pedro Antón

En el artículo anterior, se vio como tratar un sprite de forma un tanto peculiar. Recordemos que era sometido a una matriz de transformación y se consigue el efecto de la lente. Este artículo, aunque técnicamente sencillo, será de gran ayuda a todos aquellos lectores que deseen incluir algún tipo de sprite a sus producciones.



QUÉ ES UN SPRITE

Los sprites son gráficos de tamaño normalmente fijo, que unidos o combinados pueden crear una imagen animada.

Aunque un correcto tratamiento de sprites puede dar lugar a una imagen animada, no sólo tienen esta finalidad, además pueden formar imágenes, como en el caso del ejemplo que acompaña a este artículo. Un sprite puede ser tratado de incontables formas, la forma más común de usar un sprite es animar personajes o elementos en un juego. Aunque este no es el caso, se comentarán brevemente algunos puntos a tener en cuenta a la hora de hacer una secuencia animada tipo vídeo-juego.

Se debe tener en cuenta donde y para qué se va a usar dicho sprite, en orden a optimizar lo máximo posible el procedimiento que coloque los sprites en la zona de trabajo, léase memoria de vídeo o buffer de trabajo. Dicho de otro modo, se debe tener en cuenta si el sprite necesita tener un color transparente y las dimensiones de dicho sprite:

Como el lector habrá comprobado siempre se trabaja con números pares a la hora de hablar de dimensiones o resoluciones en el PC. Este dato, aunque aparentemente insignificante, se debe tener en cuenta a la hora de diseñar un sprite, fundamentalmente, en el ancho del sprite. Como se dijo en el primer artículo de este curso, las instrucciones de transferencia de bloques usan accesos por DMA, siendo notablemente más rápidas, que las posibilidades alternativas, en la mayoría de los casos. Por lo tanto si el ancho del sprite es par se podrá optimizar la ruti-

El manejo de los sprites, de forma original, da paso a sorprendentes posibilidades que complementarán una demo.

na que ponga dicho sprite poniendo en el registro contador CX la mitad del ancho y añadiendo la instrucción *REP MOVSW* en lugar de *REP MOVSB*. Esto hará que nuestra rutina de sprites gane el doble de velocidad, ya que los procesadores 80x86 son como mínimo de 16 bits. Mayor optimización se podría conseguir si en lugar de usar anchos pares, se usan anchos múltiplos de cuatro.

En segundo lugar los sprites que usen un color máscara se pondrán de forma diferente a los que no usen dicho color. El motivo, es sencillo, aquellos se sprites que vayan a ser usados para crear un fondo o tengan un fondo de igual color al de la máscara, se pueden poner en memoria de la forma anteriormente expuesta. De lo contrario se debe comprobar que el valor del pixel a poner no es la máscara, en cuyo caso, no será necesario escribir dicho pixel.

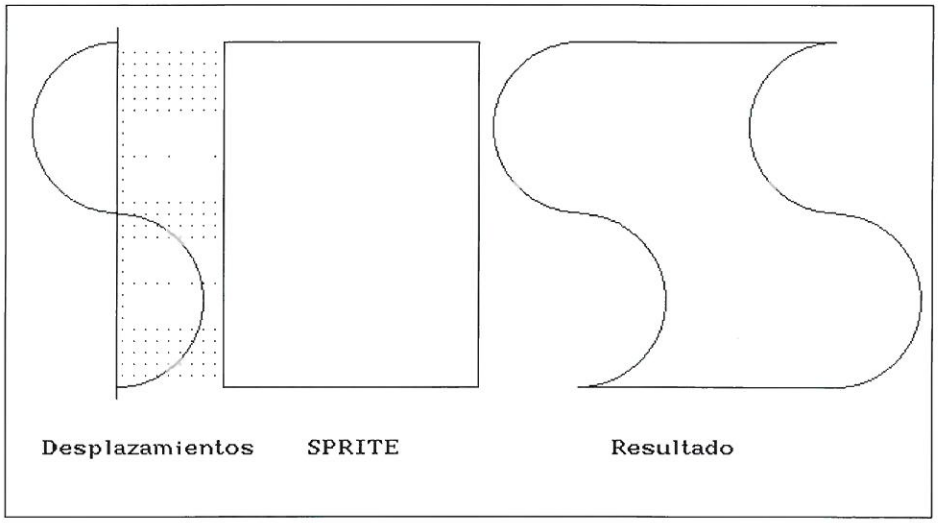
Pero además, en el caso de usar sprites para crear algo similar a un juego, será necesario incluir un parámetro con el orden a colocar dicho sprite, de esta forma, se consigue, al construir la imagen, que unos sprites se superpongan a otros.

EL RECTÁNGULO SUCIO

A la hora crear una nueva imagen es necesario restaurar el fondo.

Cuando se generan animaciones con un fondo, es necesario restaurar ese fondo, para ello, existen varios métodos que dependen directamente de la cantidad y tamaño de los sprites. Es interesante destacar el método del rectángulo sucio, particularmente útil en el caso de animar una sola figura. En el caso de que la figura sea de unas dimensiones relativamente reducidas, aproximadamente una cuarta parte de la zona de memoria a volcar, puede resultar interesante capturar el trozo de área limpia donde se va a colocar el sprite o gráfico en cuestión antes de ponerlo. De esta forma, al generar la siguiente imagen sólo es necesario restaurar esa zona del fondo, denominada rectángulo sucio, antes de actualizar los datos y volver a presentar el gráfico en cuestión.

En el ejemplo que acompaña a este artículo la zona a restaurar es de unas dimensiones bastante considerables. El método a emplear en este caso, es el de mezclar las imágenes en un buffer, que será volcado a la memoria de video después de esperar el consiguiente retrazo vertical.



TRATAMIENTO DE UN SPRITE

En una zona de memoria se pueden realizar gran cantidad de modificaciones.

En el artículo del mes pasado, se vio una forma de tratar una zona de memoria, mediante una matriz de transformación. Este mes, se verá como modificar una zona de memoria o sprite, modificando el desplazamiento original ("offset").

Este desplazamiento puede ser modificado para crear distintos efectos, por ejemplo, una fuente en cursiva, en cuyo caso bastará con aplicar un sencillo desplazamiento lineal. Pero mejor que esto es aplicar una curva que posea idéntico principio y final. Un ejemplo sencillo de esto es una curva senoidal, con ella se trabaja en el ejemplo que acompaña este artículo. Es recomendable la creación de una tabla, donde serán almacenados los desplazamientos, ahora bien, la cantidad de puntos a coger para dar forma a la onda queda a elección del programador, por supuesto, en el ejemplo la onda quedará definida por 256 valores. El porqué de este valor, ya se explicó en artículos anteriores, definir un contador de ángulos con 8 bits, evitando con esta definición comprobar si se ha llegado a 360 grados.

PutSpriteSin

```
Procedure PutSpriteSin
(   OrgSeg,OrgOff: Word;
    DesSeg,DesOff: Word;
    Ancho,Alto  : Word;
    Angulo:Word)      ;Assembler;
Asm
    push ds
    mov  ax,DesSeg
    mov  es,ax
    mov  di,DesOff
    mov  ax,OrgSeg
    mov  ds,ax
    mov  si,OrgOff
    mov  bx,di
@@OtraLinea:
    push di
    push ds
    mov  ax,Seg TblSinCos
    mov  ds,ax
    mov  di,OffSet TblSinCos
    mov  ax,Angulo
    shl  ax,1
    add  di,ax
    mov  ax,[ds:di]
    pop  ds
    pop  di
    sar  ax,5
    add  di,ax
    add  Angulo,5
    cmp  Angulo,255
    jbe  @@NoPasaNada
    mov  ax,Angulo
    sub  ax,255
    mov  Angulo,ax
@@NoPasaNada:
    mov  cx,Ancho
@@OtroPixel:
    mov  al,ds:[si]
    cmp  al,0
    je          @@Mask
    mov  es:[di],al
@@Mask:
    inc  si
    inc  di
    dec  cx
    jnz  @@OtroPixel
    add  bx,320
    mov  di,bx
    cmp  di,64000
    ja          @@Fin
    add  si,320
    sub  si,Ancho
    dec  Alto
    jnz  @@OtraLinea
@@Fin:
    pop  ds
End;
```

El procedimiento *PutSpriteSin* trabaja de la siguiente forma: Necesita la dirección (segmento y offset) de donde se encuentra el sprite a poner y la dirección donde se va a poner el sprite. Así como el ancho, el alto y el ángulo donde se desea empezar a aplicar el seno. Una vez colocados los segmentos y punteros ES:DI y DS:SI en sus correspondientes direcciones, se procede a ir poniendo los datos del sprite, pero desplazando el valor del seno para el ángulo dado. En este caso, sólo es necesario modificar el desplazamiento, una vez por fila. Como en el ejemplo adjunto se usa un fondo, la rutina usa el color cero como máscara, por lo tanto antes de poner el valor de origen, en el correspondiente destino, es necesario comprobar que dicho valor no es cero, en cuyo caso, se actualizan todos los contadores, pero el dato no se pone. Los contadores que usa la rutina son: ancho, alto, offset de origen, offset de destino y el ángulo de la curva, este último valor se incrementa de cinco en cinco, para conseguir un movimiento alegre de los sprites. Es interesante destacar la parte de la rutina que comprueba si el offset a poner es mayor de 64000, el motivo de esta comprobación es, evidentemente, hacer que el sprite salga por la parte inferior de la pantalla.

EL BUFFER

El buffer se puede usar para hacer un scroll.

La VGA, en el modo de trabajo usado en este curso, tiene una un ancho de 320 pixels. Generalmente no se ha escrito directamente en memoria de vídeo a lo largo del curso. Reservando un buffer que sea tratado con un ancho lógico de dos o tres veces el ancho físico de nuestro modo de trabajo, se pueden escribir el doble o el triple de sprites de los que se representarían normalmente en pantalla. Si dicho buffer es mostrado en porciones de memoria iguales al ancho físico de la VGA se conseguirá un scroll software. Este tipo de scrolls, no suelen ser muy rápidos en ordenadores normales, pero la idea puede servir para que el lector ponga en práctica algunos de los conocimientos adquiridos a lo largo del curso.

Rutinas de sprites

```
PROCEDURE CopySprite
(   SegOrg,OffOrg,
SegDes,OffDes, XDim,YDim:Word);assembler;
Asm
    push ds
    mov  ax,SegOrg
    mov  ds,ax
    mov  si,OffOrg
    mov  ax,SegDes
    mov  es,ax
    mov  di,OffDes
    mov  dx,LogicalWide
    sub  dx,XDim
    mov  bx,YDim
@@MoreY:
    mov  cx,XDim
    rep  movsb
    add  si,dx
    dec  bx
    jnz  @@MoreY
    pop  ds
End;
PROCEDURE PutSpriteClip
(   SegOrg,OffOrg,SegDes:Word;
    XDes,YDes:Integer;
    XDim,YDim,
X1Clip,Y1Clip, X2Clip,Y2Clip:Word);assembler;
Asm
    mov  ax,X1Clip
    mov  bx,XDes
    add  bx,XDim
    cmp  bx,ax
    jle  @@End
    mov  ax,Y1Clip
    mov  bx,YDes
    add  bx,YDim
    cmp  bx,ax
    jle  @@End
    mov  ax,X2Clip
    cmp  XDes,ax
    jge  @@End
    mov  ax,Y2Clip
    cmp  YDes,ax
    jge  @@End
    push ds
    mov  ax,SegOrg
    mov  ds,ax
    mov  si,OffOrg
    mov  ax,SegDes
    mov  es,ax
    mov  ax,XDes
    cmp  ax,X1Clip
    jg          @@NoClipIzqdo
    mov  ax,X1Clip
    sub  ax,XDes
    add  si,ax
    mov  ax,XDes
    add  ax,XDim
    sub  ax,X1Clip
    mov  XDim,ax
    mov  ax,X1Clip

    mov  XDes,ax
@@NoClipIzqdo:
    mov  ax,YDes
    cmp  ax,Y1Clip
    jg          @@NoClipSup
    mov  ax,Y1Clip
    sub  ax,YDes
    mov  bx,LogicalWide
    imul bx
    add  si,ax
    mov  ax,YDes
    add  ax,YDim
```

```
    sub  ax,Y1Clip
    mov  YDim,ax
    mov  ax,Y1Clip
    mov  YDes,ax
@@NoClipSup:
mov  ax,LogicalWide  { Calcula el offset }
    mov  bx,YDes
    imul bx
    mov  di,ax
    add  di,XDes
    mov  ax,XDes
    add  ax,XDim
    cmp  ax,X2Clip
    jl          @@NoClipDrcho
    mov  ax,X2Clip
    sub  ax,XDes
    mov  XDim,ax
@@NoClipDrcho:
    mov  ax,YDes
    add  ax,YDim
    cmp  ax,Y2Clip
    jl          @@NoClipInf
    mov  ax,Y2Clip
    sub  ax,YDes
    mov  YDim,ax
@@NoClipInf:
mov  cx,LogicalWide  { Cte de cambio de linea }
    sub  cx,XDim
@@AnotherLine:
    mov  dx,XDim
@@AnotherPix:
    mov  al,ds:[si]
    cmp  al,0
    je          @@NoPutPix
    mov  es:[di],al
@@NoPutPix:
    inc  di
    inc  si
    dec  dx
    jnz  @@AnotherPix
    add  di,cx
    add  si,cx
    dec  YDim
    jnz  @@AnotherLine
    pop  ds
@@End:
End;
```

UN SCROLL

Recortando los sprites de forma correcta se consigue un scroll.

Se puede conseguir un scroll, sin usar el buffer, ni los puertos de la VGA. Se puede conseguir, recortando los sprites en una ventana. Tampoco es una buena forma de hacer un scroll, pero recortar los sprites en ventana, es realmente útil, además de recomendable. El procedimiento *PutSpriteClip* se encarga de ello, recorta los sprites en la ventana indicada.

LA UNIDAD DEMOVGA

La unidad de este curso continua creciendo.

En este artículo tres nuevas funciones han sido añadidas a la unidad, cuya finalidad es copiar un sprite de una zona de memoria a otra. *PutSprite* y *CopySprite* realizan, casi, el mismo trabajo. La diferencia entre ellas estriba en la existencia del color cero como máscara, o no. El lector podrá usar otro color como máscara, con ligeras modificaciones, o incluso pasar el color máscara por parámetro. Notesé que la inclusión de un color máscara, hace la rutina de poner sprites, bastante más lenta.

Las fuentes han de tener iguales dimensiones para todos los caracteres

Lo más destacable de estas rutinas es que el origen y el destino deben tener igual ancho lógico e igual a la constante *LogicalWide*. El motivo de crear esta constante es simplificar el código para que el lector comprenda mejor el funcionamiento de las rutinas, así como facilitar el desarrollo de nuevas rutinas adaptadas al código del lector.

El tercero de los procedimientos que trata sprites, es el más útil, puesto que se trata del mismo *PutSprite* pero recortando en ventana, su nombre, *PutSpriteClip*. Este procedimiento necesita las coordenadas que definen la ventana, y las coordenadas (x,y) del sprite a poner, así como la dirección de memoria donde se encuentra el sprite y la dirección donde queremos colocarlo, también se le debe indicar el ancho y el alto del sprite. El procedimiento ajusta el ancho, el alto y las coordenadas (x,y) para recortar el sprite. El código puede ser ligeramente optimizado, consiguiendo así una mejor eficacia, pero no he considerado conveniente hacerlo para que la rutina gane en claridad y comprensión. Se debe mencionar que la rutina no comprueba si las coordenadas están en el orden correcto, así pues, los parámetros (X1Clip, Y1Clip) y (X2Clip, Y2Clip) se corresponden con los puntos que definen la ventana donde se recortará el sprite, esto es, la esquina superior izquierda, y la esquina inferior derecha.

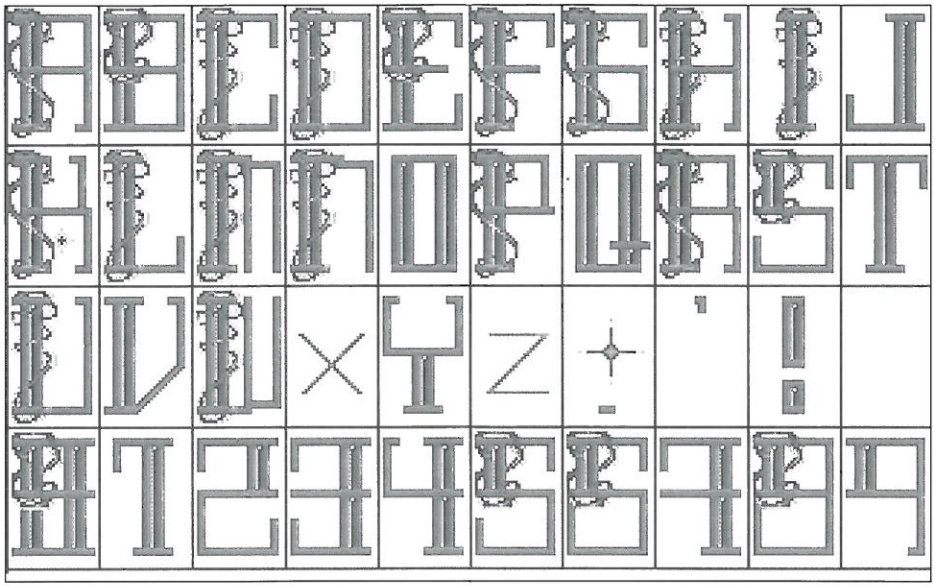
En una zona de memoria se pueden realizar gran cantidad de modificaciones

Además se ha añadido una función cuya finalidad es calcular el offset de un par de coordenadas (x,y) llamado Mcga2Off. Esta función únicamente trabaja en resolución 320x200, la importancia de ella reside en la velocidad de proceso. Para calcular el offset de un par de coordenadas (x,y) en unas dimensiones XDim, YDim se debe multiplicar la coordenada y por YDim y al resultado sumarle la coordenada x, sin embargo la función *Mcga2Off* no usa la instrucción de multiplicar debido la cantidad de ciclos de CPU que consume. Pero, aunque esta función es rápida, una rutina rápida deberá incluir esos cálculos dentro de su propio código, evitando de esta manera hacer un CALL, pues de lo contrario, los ciclos que son ahorrados en el cálculo del offset se consumen en el *CALL*.

FUENTES CON SPRITES

Si los sprites son textos, se pueden crear fuentes gráficas.

Como se puede ver en el ejemplo adjunto, la correcta combinación de los sprites puede dar lugar a frases con sentido. Tratando correctamente los sprites es posible crear rutinas de fuentes gráficas, para ello se deben tener en cuenta varios factores:



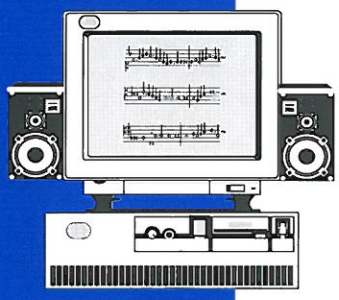
Las fuentes han de tener iguales dimensiones para todos los caracteres.

Los offsets de cada letra se precalcularán al comienzo de la ejecución del programa, o bien se almacenarán como constantes.

A la hora de diseñar la fuente se debe tener en cuenta la posibilidad de cambiar el color base de esta, para ello no se deben emplear muchos colores en el diseño de la fuente, y los que se empleen deberán estar correlativos.

Teniendo en cuenta estos factores y empleando los conocimientos adquiridos en este curso, al lector no le será difícil crear su propio "interface" de fuentes gráficas.

CÓMO UTILIZAR EL REPRODUCTOR MUSICAL CON UNA DEMO



Héctor Martínez

Hasta el momento se ha explicado como hacer un reproductor musical, pero sería muy pobre utilizar el player tal y como lo tenemos ahora, ya que no aporta ninguna sincronización al programa principal (la demo). En este artículo se verá como sincronizar la música con el programa que está corriendo en primer plano.

¿POR QUÉ HAY QUE SINCRONIZARSE?

Tal y como tenemos el reproductor musical hasta el momento, sirve para reproducir piezas musicales muy variadas, pero normalmente, cualquier programa en el que corre un player de MOD's, S3M's u otro formato musical, viene asociado a un juego, gráficos, demos, etc.

Si simplemente utilizamos el player en un programa, mientras hacemos cualquier otra cosa en el programa que corre en primer plano, dará la impresión de que cada cosa va por un lado, quedando independizadas totalmente lo que es la música con la aplicación en cuestión. De hecho, esto mismo se podría conseguir en un sistema multitarea abriendo en una tarea el reproductor musical y en otra la aplicación que queremos ejecutar con música de fondo.

La idea de este artículo es, por lo tanto, ligar de alguna manera la aplicación que corre en primer plano con la música en cuestión.

IDEAS BÁSICAS PARA SINCRONIZARSE

Si nos paramos a pensar de que manera nos podríamos sincronizar con la aplicación que corre en primer plano, podríamos pensar en las siguientes:

- Sincronización con el cambio de un patrón de la composición: Casi todas las composiciones musicales, están hechas de tal manera que cada patrón corresponde a un segmento de la composición musical que parece tener muchas veces su principio y su final; de manera que los cambios de tema siempre vienen reflejados al final de un patón o *pattern*.

- Sincronización con un instrumento de la canción: Muy útil cuando queremos que ocurra un acontecimiento al ritmo de un instrumento musical. Normalmente se utiliza con instrumentos de percusión, ya que son de corta duración.

- Sincronización con un cierto punto de la música: Se utiliza cuando en un cierto momento, en cualquier punto de la canción, que no se corresponde con un cambio de patrón, ni con

Las demos más espectaculares están perfectamente sincronizadas con la música

En esta serie de artículos, se ha estado siempre haciendo referencia a las demos, de hecho, cualquier demo que se precie tiene asociada a ella una composición musical polifónica de instrumentos digitalizados.

el sonido concreto de un instrumento, queremos que ocurra cierto acontecimiento. La manera mas ingeniosa de hacer esto, es utilizando un efecto de los que no se utilizan para sincronizar. En los MOD's, se utiliza el efecto número 8, ya que carece de significado. Esta idea, es la que utilizaron los hermanos Arévalo, en su player para demos llamado Demo VT.

SINCRONIZACIÓN CON EL CAMBIO DE PATRÓN

Para implementar esta sincronización, basta con añadir al player, una función que devuelva cierto, en el momento en que hemos rebasado ese patrón. En este tipo de sincronización, normalmente no nos interesa el momento justo en el que tiene lugar el acontecimiento, sino simplemente, nos interesa saber si ha pasado o no. El algoritmo en pascal de esta sincronización sería:

```
function EsperaPattern(Song: TSong;
n:Word):Boolean;
(* n indica el patrón a partir del cual
nos tiene que devolver falso *)
 begin
        if Song.SeqPos<n then
EsperaPattern:=True (* return true *)
   else EsperaPattern:=False; (* return
false *)
 end;
```

Esta función no bloque la ejecución del programa, simplemente se utiliza para consultar. Devolverá cierto mientras no hallamos llegado al punto de sincronización. Suponiendo que lo que nos interese sea estar ejecutando una rutina hasta que la música llegue al patrón número 3; la manera de utilizarla desde el programa principal podría ser la siguiente:

```
while EsperaPattern (3) do
  (* rutina que corre en primer plano
hasta el patrón 3º *)
```

SINCRONIZACIÓN CON UN INSTRUMENTO DE LA CANCIÓN

Recordemos, que en el player, tenemos un array de canales. Por lo tanto, el instrumento con el que nos queremos sincronizar, podría sonar por cualquier canal. En este tipo de sincronización, nos interesa el momento exacto en el que está sonando el instrumento, ya que normalmente suena muchas veces el mismo instrumento, y lo que nos interesa es que ocurra un acontecimiento cada vez que suena el instrumento. Esto implica que haya que hacer esta consulta muy rápido, ya que al mínimo retraso, nos podemos saltar alguno de los acontecimientos. Para no perder tiempo buscando por que canal está sonando el instrumento, sincronizaremos por instrumento y canal. Es decir, esperaremos a que suene un cierto instrumento por un canal dado. La desventaja de esto, es que se requiere de un conocimiento previo de la composición musical. También habrá que componerla teniendo en cuenta esto último, y por lo tanto hay que intentar que el instrumento con el que nos queremos sincronizar esté siempre en el mismo canal, o en el caso de estar en dos habrá que llamar a la función de sincronizar dos veces. El algoritmo de esta función sería:

```
function EsperaInstrumento(Song:
TSong;canal,Inst:Word):Boolean;
var
 Note:TNote;
begin
 SongGetNote(Song,canal,Note);
 if Note.InstNum=Inst then
   EsperaInstrumento:=True (* return
true *)
 else
   EsperaInstrumento:=False; (* return
false *)
end;
```

Esta función devolverá cierto en el momento en que está sonando cierto instrumento por el canal especificado. La típica manera de utilizarla será para ejecutar siempre la misma rutina (cambio de paleta, dibujado y borrado rápido de un objeto, ...). Lo haremos de la siguiente manera:

```
while not Fin do
begin
  (* rutina que corre en primer plano
*)
   if EsperaInstrumento(3,5) then
CambiarPaleta (* por ejemplo *)
end;
```

SINCRONIZACIÓN MEDIANTE UN EFECTO NO UTILIZADO

Este sistema permite la sincronización en cualquier punto de la composición. La desventaja que tiene es que a la hora de componer la música, tenemos que colocar en la partitura los puntos de sincronismo. En el caso del MOD cogeríamos el efecto 8, y en el caso del S3M, el mas indicado es el 'X', que representa al efecto 8 del MOD. A partir de esta técnica podemos tomar dos posturas. Una sería por *polling* o encuesta, al igual que los tipos de sincronizaciones explicados hasta ahora, y la otra por interrupción. En el caso que he tratado yo, están contempladas las dos a la vez.

Este sistema permite la sincronización en cualquier punto de la composición

- *Por encuesta:*
  Al tratar el efecto en cuestión, se debe actualizar una cierta variable booleana para indicar que el punto de sincronización ya ha pasado. La función de consulta se tiene que encargar de volver a cambiar el valor de esta variable.

- *Por interrupción:*
  No es que se trate puramente de una interrupción, pero el efecto producido es similar al de las interrupciones. Primero habrá que indicarle al reproductor la dirección de la rutina de atención a esta "interrupción". El reproductor, automáticamente llamará a la rutina en cuestión. Hay que tener cuidado de que el código a ejecutar en la rutina de atención a la interrupción no

Rutinas para el manejo del DAC

```
UNIT DAC;
{
  Rutinas para el manejo del DAC casero, con soporte de doble buffer virtual.
  Por Luis Crespo, FidoNet 2:343/108.21, Internet d8089110@est.fib.upc.es
}

INTERFACE

USES
  Hardware;

  PROCEDURE DACInit;
  PROCEDURE LPT1DAC(b:Byte);
  PROCEDURE LPT2DAC(b:Byte);
  PROCEDURE DACPlayBuf(pBuffer:Pointer; BufferSize:Word; VAR Frec:Word; BufProc:TBufProc;
LPTNum:Byte);
  PROCEDURE DACStopBuf;


IMPLEMENTATION

USES
  DOS;

VAR
  OldTimerISR : Pointer;  {Puntero a la antigua ISR del timer}
  pBufDAC     : PByte;    {Puntero al buffer de DMA falso}
  BufSize,                {Tamaño del buffer}
  BufPos      : Word;     {Offset de la muestra en el buffer}
  NextBuffer  : TBufProc; {Rutina de usuario}

  DACOut      : PROCEDURE (b:Byte);

CONST
  LPT1Addr:Word=$378;
  LPT2Addr:Word=$278;

  PROCEDURE DACInit;
  BEGIN
    IF MemW[$40:$08]<>0 THEN LPT1Addr:=MemW[$40:$08];
    IF MemW[$40:$0A]<>0 THEN LPT2Addr:=MemW[$40:$0A];
  END;

  PROCEDURE LPT1DAC(b:Byte);
  BEGIN
    Port[LPT1Addr]:=b;
  END;

  PROCEDURE LPT2DAC(b:Byte);
  BEGIN
    Port[LPT2Addr]:=b;
  END;


  PROCEDURE TimerISR; INTERRUPT;
  TYPE
    TBarr=ARRAY [0..0] OF Byte;
  BEGIN
    EOI;
    ASM STI END;
    DACOut(TBarr(pBufDAC^)[BufPos]);
    Inc(BufPos);
    IF BufPos=BufSize THEN NextBuffer;
    IF BufPos=BufSize*2 THEN
    BEGIN
      BufPos:=0;
      NextBuffer;
    END;
  END;

  PROCEDURE DACPlayBuf(pBuffer:Pointer; BufferSize:Word; VAR Frec:Word; BufProc:TBufProc;
LPTNum:Byte);
  BEGIN
    pBufDAC:=pBuffer;
    BufSize:=BufferSize;
    NextBuffer:=BufProc;
    BufPos:=0;
    IF LPTNum=1 THEN DACOut:=LPT1DAC ELSE DACOut:=LPT2DAC;
    GetIntVec(8,OldTimerISR);
    SetIntVec(8,@TimerISR);
    SetTimerFrec(Frec);
  END;

  PROCEDURE DACStopBuf;
  BEGIN
    ProgTimer0(0);
    SetIntVec(8,OldTimerISR);
  END;


END.
```

sea demasiado lento, ya que podría llegar a parar el reproductor.

La forma de utilizar la sincronización por interrupción sería la misma que la de cambio de patrón, sin embargo, la de interrupciones es bastante diferente a las otras, ya que el programa principal no se tiene que preocupar del momento en el que llegue el punto de sincronismo, ya que tan pronto lleguemos, se ejecutará nuestra rutina. Lo único que habrá que hacer es empezar inicializando la rutina de atención a la "interrupción".

EL PROGRAMA DE EJEMPLO

El programa de ejemplo que se suministra con este artículo utiliza algunas de estas sincronizaciones como prueba de su utilidad. Utiliza el cambio de patrón para pasar de una fase del programa a otra. En la primera fase, sale un contador que cuenta hacia adelante. Al llegar un cierto patrón, comienza a decrementar en lugar de incrementar.

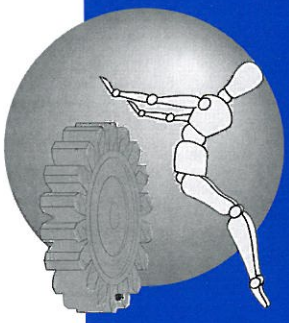
A la vez que se va haciendo esto, cada vez que llega un cierto instrumento aparecerán letras por la pantalla. Realmente el ejemplo no es nada espectacular, pero ayuda a ver la utilidad de la sincronización.

Como ejemplo de una demo que se basa en la sincronización de la música, tenemos una de las que participaron en la Euskal Party 95, la que quedó en tercer puesto, llamada Fistropía, utiliza este tipo de sincronización. La demo en sí no es muy espectacular, pero está perfectamente sincronizada con la música. Se puede apreciar como utiliza la sincronización por cambio de patrón entre fase y fase de la demo, y como utiliza la sincronización por instrumento mientras salen las frutas y los muñecos.

Contactar con el autor:
Internet d8128595@est.fib.upc.es

RAY TRACING (V): TEXTURAS

Álvaro Silgado



Durante todo este curso, se ha podido ir descubriendo las técnicas principales que todo programa de Ray Tracing debe contemplar. Los artículos que se van a ir viendo a partir de ahora seguirán el objetivo de obtener una mayor calidad, bien sea mediante la incorporación de nuevos objetos, bien mediante técnicas de realismo.

En este capítulo se va a estudiar una de las mejores técnicas para proporcionar realismo a una escena : las texturas.

ENVOLTORIOS DE OBJETOS

El concepto básico para entender cómo se implementan las texturas es el de "envolver". Lo que se va a hacer es ni más ni menos que eso : envolver los objetos con "papeles de colores" como si de caramelos se tratara. Con este proceso lo que se logra es que el color del objeto varíe a lo largo de su superficie, según el mapa que define la textura. Para ello, se suele usar como envoltorio una fotografía del material del que se quiere diseñar el objeto. Este es el tipo de textura más elemental.

Además de poder definir diferentes colores en los objetos con los *mapas de color*, con las texturas se puede ir aún más allá. Así, es posible modificar su rugosidad mediante *mapas de rugosidad*, variar su capacidad de reflexión usando *mapas de reflexión*, hacer el objeto más o menos transparente por zonas definiendo *mapas de opacidad* y forzar que los rayos que trazan la escena no hagan intersección en determinados puntos mediante *mapas de intersección*.

Dada su similitud, en este capítulo se van a estudiar los mapas de color y los mapas de intersección. Sea cual sea el tipo de textura a implementar, el proceso típico que se sigue viene a ser el siguiente :

- Reservar una zona de memoria lo suficientemente grande como para albergar el mapa que se va a utilizar para proporcionar la textura al objeto. Generalmente, y a menos que se quiera obtener un nivel de detalle muy alto, se suelen utilizar imágenes de 64 Kb como mucho, ya que son más fáciles de manipular al utilizarse un único segmento de memoria y proporcionan un gran realismo.
- Definir una *función de envoltura* en la clase del objeto. A esta función se le pasará un punto del objeto, el vector normal en dicho punto y su material, y devolverá dos coordenadas (U y V) comprendidas entre 0 y 1. Estas coordenadas se conocen con el nombre de *coordenadas de proyección*, y se van a utilizar para obtener el pixel asociado a dicho punto en la imagen o *mapa*. Por lo tanto, son las que deciden el color (o rugosidad, opacidad, etc.) que va a tener el objeto en dicho punto.
- Al hacer intersección un rayo con un objeto, comprobar si dicho objeto posee algún tipo de mapa de los mencionados anteriormente. En caso afirmativo, obtener los valores de U y V en el punto de intersección.
- Con estos valores, se accede al mapa y obtener el valor del pixel correspondiente.
- Una vez obtenido el valor del mapa en el punto asociado del objeto, se alterará su superficie, devolviendo un color diferente, o una superficie diferente, o una normal diferente, según el tipo de textura que se esté implementando. O incluso diciendo que el rayo no ha hecho intersección con dicho objeto en ese punto.

TEXTURAS Y PATRONES

Si se repasa un poco los anteriores artículos, sobre todo el primero, se puede recordar una herramienta muy similar a las texturas, que se ha venido utilizado en

Con el uso de las texturas es posible crear objetos de una complejidad sorprendente con muy poco esfuerzo. es un nuevo paso hacia la reproducción de la realidad.

toda esta serie de artículos : *los patrones*.. Mediante el uso de patrones se ha podido ir haciendo diseños de colores, y también se podrían haber usado para modificar opacidad, reflexión, etc. Pero, ¿ qué diferencia hay entre las texturas y los patrones ?

En realidad, el concepto es el mismo. La única diferencia reside en la forma en la que se decide el color (o cualquier otra cualidad) a asignar a cada punto de la superficie. Los patrones son funciones independientes del tipo de objeto al que se está aplicando. O dicho de otra forma: no conocen la geometría del objeto al que hacen referencia. La decisión del tipo de material a asignar está en función de las coordenadas del punto que se está estudiando en ese momento. Pero dentro de la función del patrón, no se sabe qué tipo de objeto es. Por lo tanto, para poder aplicar un mismo diseño de colores en, por ejemplo, una esfera y en un polígono, será necesario diseñar un patrón para cada objeto. Por otro lado, las texturas trabajan en todo momento con funciones de envoltura, que hacen que se pueda recorrer (o aplicar) un mapa de manera uniforme sobre cualquier objeto, con independencia de la geometría de éste. De esta forma, es muy sencillo aplicar una foto de, por ejemplo, un trozo de mármol sobre una esfera o sobre un polígono y obtener en ambos casos un resultado similar. Pero es más complicado hacer motivos, por ejemplo, a rayas del mismo tamaño. Si se emplea una textura con rayas y se aplica sobre un objeto no plano, la textura quedará deformada obteniéndose unas franjas más gruesas que otras.

¿Qué es lo que interesa usar, texturas o patrones? La respuesta es : ambos. En general, se utilizarán patrones siempre que el motivo o la variación de material dependa de las coordenadas en las que se encuentra el objeto, y se emplearán texturas cuando dicha variación dependa de la geometría intrínseca del objeto. De echo, se podrán usar ambas cosas a la vez. Así, será posible definir dos materiales, cada uno con una textura diferente y elegir uno u otro según un patrón.

LA "CLASE DE MAPA" QUE SE NECESITA

Para poder almacenar los mapas de una forma sencilla, y dentro de la filosofía del programa orientado a objeto, se ha diseñado la clase *mapa*. En esta clase hay cuatro métodos, todos ellos muy básicos.

- El primero de los métodos, es el constructor, al cual, se le pasa un fichero de imagen PCX con 256 colores que es cargado en memoria, siempre que no ocupe más de 64 Kb. La paleta de la imagen también es almacenada. De esta forma, se puede obtener el índice del pixel o su color. Si la imagen es superior a 64 Kb o no hay memoria, el proceso de construcción de la clase falla.
- El segundo de los métodos es el destructor de la clase, que libera toda la memoria reservada por el constructor.
- El tercer método, *obtener_indice*, es el que más se va a usar. Se le pasan dos parámetros, U y V, que son el resultado de aplicar una función de envoltura en un punto de un objeto, y devuelve un índice (entre 0 y 255). Con el índice se podrá consultar el color de la paleta asociada o tomar decisiones directamente con este valor.
- El cuarto, *obtener_color*, devuelve el color real en la paleta que le corresponde a un índice determinado.

Con esta definición de clase, se puede crear una única vez un material y aplicárselo a diferentes objetos. Además, se ha añadido un puntero a otro mapa, para poder encadenarlos y liberarlos fácilmente. De esta forma, el destructor de la clase mundo va recorriendo uno por uno todos los mapas que se han creado y va llamado a su destructor.

CÓMO CAMBIAR EL COLOR DE UN OBJETO

A partir de ahora, hay dos cambios importantes. Por un lado, todos los objetos tienen una función de envoltura, llamada *envolver*. Como se ha dicho antes, a esta función se le pasa un punto, el vector normal en dicho punto y su material, y devuelve las coordenadas U y V asociadas a dicho punto.

Por otro lado, la clase *material* se ha modificado para que admita mapas. De esta forma, además de todos los campos anteriores, contiene dos punteros a la clase mapa :

- **color** : define el mapa de color del objeto.
- **intersec** : define el mapa de colisión o mapa de intersección del objeto.

También se han añadido variables de control de textura, como *espejo*, *escala_u* y *escala_v*, así como ciertas variables auxiliares. Estas variables se explican un poco mas adelante.

Además, los constructores de la clase material se han eliminado, dejando los constructores por defecto, ya que el número de variables ha aumen-

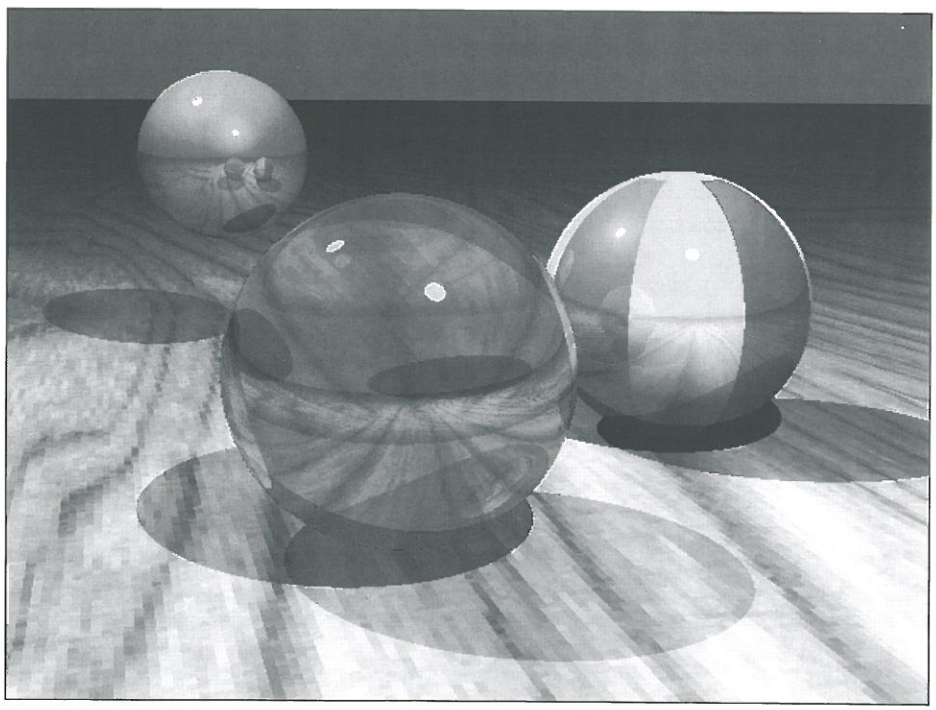


Figura 1. Las texturas hacen que esta imagen sea realmente espectacular.

tado tanto que merece más la pena definir una por una, una vez creado el material.

Para poder implementar esta nueva cualidad de los objetos, basta con sustituir en la función *iluminar* todas las referencias al campo *difusa* por una llamada a la función envoltorio, para posteriormente obtener el color del mapa en las coordenadas U y V obtenidas. Esto se hará siempre y cuando el puntero *color* sea distinto de NULL (en cuyo caso se procedería como hasta ahora).

De esta forma, se van a ir obteniendo diferentes colores a medida que se recorre el objeto. Un ejemplo de esta técnica se puede observar en la figura 1.

CÓMO NO HACER INTERSECCIÓN CON UN OBJETO

Para implementar un mapa de intersección es necesario retocar las funciones de intersección de cada uno de los objetos. De lo que se trata es de "engañar" al trazador de rayos, para que crea que no se produce intersección en puntos en donde realmente sí se produce.

Así, cuando se produce una intersección con un objeto, antes de que la función diga que sí se ha producido dicha intersección, se obtendrán las coordenadas U y V asociadas al punto de intersección, se consultará en el mapa el píxel correspondiente a dichas coordenadas y si su valor es 0, entonces la función dirá que no se ha producido intersección en dicho punto. Por supuesto, todo esto se hace siempre y cuando el objeto tenga definido un mapa de intersección. En caso contrario, se procedería como de costumbre.

Hay que tener una precaución importante en objetos que puedan tener más de una intersección con un mismo rayo. Generalmente, las funciones de intersección de dichos objetos (como la esfera) se optimizan para que devuelvan directamente la intersección más próxima al origen del rayo. En caso de que uno de estos objetos tenga un mapa definido, será necesario ir estudiando punto por punto todas las intersecciones del rayo con dicho objeto, desde el más próximo al origen del rayo hasta el más alejado. Para cada punto, se comprobará si, según el mapa de textura, se ha producido colisión o no. En caso afirmativo, se devuelve dicho punto y se ha terminado. En caso contrario, se procede de igual manera con el siguiente punto.

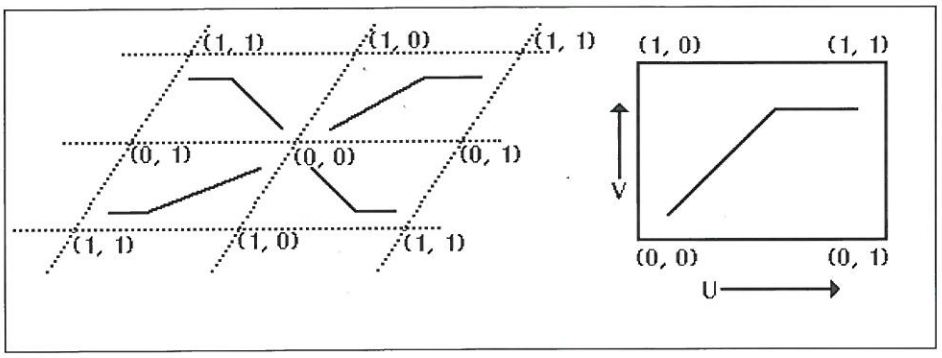


Figura 2. Aplicación de una textura al suelo.

Esta técnica va a permitir resultados sorprendentes y objetos que necesitarían técnicas muy complicadas o miles de polígonos para implementarlos. Por ejemplo, si se define un mapa de colisión formado por franjas verticales y se aplica sobre una esfera, el resultado es una esfera formada por una especie de "gajos". Definir una semiesfera con esta técnica es muy fácil. Basta con hacer un mapa de colisión de la mitad para arriba con unos y de la mitad para abajo con ceros.

Las posibilidades son infinitas, sólo se necesita un poco de imaginación.

CÓMO ENVOLVER EL SUELO

Tanto en el caso del suelo, como en el caso del cielo, la técnica que se va a emplear es de repetición de texturas. Esto es debido a que son objetos infinitos, y no se puede utilizar una pequeña textura para un objeto tan grande, a menos que quede limitado por otros elementos (como por ejemplo, paredes).

Para entender cómo se va a "empapelar" el suelo, se puede pensar en un parqué. En un suelo terminado en parqué, hay un motivo que se va repitiendo constantemente (las franjas horizontales y verticales de madera). El programa va a hacer lo mismo y, partiendo de un motivo repetitivo, lo irá extendiendo por toda la superficie.

Hay dos elementos que van a condicionar el acabado obtenido. Por un lado, está el factor de escala, que hará que la textura sea más grande o más pequeña con respecto al suelo. Así, por ejemplo, un factor de escala 1 en X, hará que el largo del mapa entre en una unidad de medida del suelo. El ancho del mapa también tendrá su factor de escala.

Otro de los elementos que va a influir es la forma en que se va a producir la repetición de las texturas. Si, por ejemplo, se emplea una foto de una madera, al colocar una a continuación de la otra, se va a notar los puntos de unión entre ambas. Pero si la siguiente foto es igual pero invertida de derechas a izquierdas, esta unión no se notará, ya que empezará justo donde terminó la anterior. Lo mismo ocurre con las texturas situadas encima y debajo : se pueden invertir para que coincidan perfectamente. Esta técnica se llama *repetición con espejo*. En la figura 2 se puede ver una textura con un motivo muy sencillo. Al aplicarlo sobre el suelo, automáticamente se va dando la vuelta a la textura para que los puntos de unión sean comunes.

La repetición por espejo no invierte realmente los mapas, sino que va devolviendo las variables U y V al derecho y al revés alternativamente, por lo que no se consume tiempo adicional. Para evitar que esto se produzca, basta con poner la variable *espejo* a 0 en la clase suelo. De esta forma, la textura empleada siempre se verá del derecho, una a continuación de la otra.

Las variables *escala_u*, *escala_v* y *espejo* se han implementado en la clase material, para que puedan ser usadas en cualquier objeto. Además, se han añadido variables auxiliares que acelerarán los cálculos. Para iniciar estas variables auxiliares, hay que ejecutar la función *material : : auxiliares ( )*, una vez iniciadas el resto de las variables.

Para obtener U y V, hay que fijarse en las coordenadas X y Z del punto de intersección que se está estudiando (recordemos que Y representa la altura y estamos hablando del suelo, en el

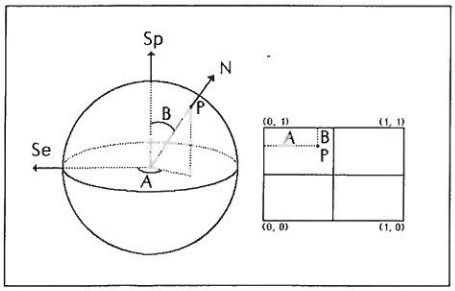


Figura 3. Aplicación de una textura a una esfera.

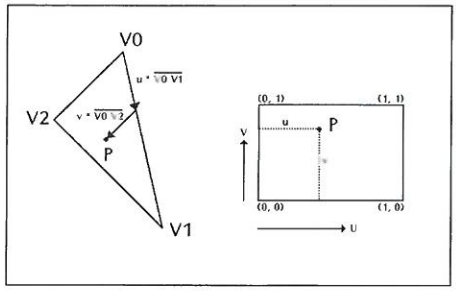


Figura 4. Aplicación de una textura a un triángulo.

cual, Y = 0). En realidad, U va a estar en función de X y V en función de Y, es decir,

U = f (x)
V = g (z)

Primero se estudiará la variable U, ya que V es muy similar. Como se ha dicho antes, el ancho de la textura va a ocupar en el suelo el número de unidades indicado por la variable escala. Así, cuando X vale 0, U vale 0, y cuando X vale *escala*, U vale 1. Por lo tanto, una primera aproximación de la función f sería :

U = f (x) = x / escala_u

Lo cual, cumple lo anteriormente dicho. Pero hay más restricciones a hacer. U no puede ser mayor que 1, así que si X sigue creciendo U también lo hace. Aquí es donde se aplica el concepto de repetición de texturas. Lo que hay que hacer es despreciar la parte entera de la división anterior y quedarse con los decimales. Así, utilizando la librería *math*,

U = modf (x / escala_u, & i) ;

Ya sólo queda tener en cuenta qué pasa si el resultado es negativo. Muy simple, se suma 1 y ya está :

U = modf (x / escala_u, & i) ;
if (U < 0) U = U + 1.0 ;

Para hacer la repetición por espejo, se trata el caso por separado. En este caso, se van a tratar las texturas como si fueran el doble de grandes, para después alterar el valor obtenido si sobrepasa el valor máximo 1. De esta forma, el cálculo de U para el suelo queda de la siguiente forma :

```
if ( espejo )
{
  U = 2 * modf ( x / ( escala_u * 2 ), &
i ) ;
  if ( U < 0 ) U = - U ;
  if ( U > 1 ) U = 2 - U ;
}
else
{
  U = modf ( x / escala_u, & i ) ;
  if ( U < 0 ) U = U + 1.0 ;
}
```

Se procede de igual manera para el cálculo de la variable V, pero usando la variable Z. Para acelerar los cálculos, se precalculan los valores 1 / escala_u, 1 / (escala_u * 2), 1 / escala_v, 1 / (escala_v * 2), para poder multiplicar en lugar de dividir.

CÓMO ENVOLVER EL CIELO

La forma más simple de envolver o *mapear* el cielo es tratarlo de igual manera que un suelo, pero situado a una altura determinada. Lo primero es comprobar que el rayo hace intersección con el cielo, lo cual, siempre va a suceder si el componente Y del vector director del rayo es positivo. En otras palabras, el rayo hace intersección con el cielo si apunta para arriba.

Una vez comprobado esto, se obtiene el punto de intersección con el rayo, el cual se produce a una distancia

d = (altura - r.pos.y) / (r.y + EPSILON)

Posteriormente, se llama a la función obtener_punto pasándole la distancia obtenida y con el punto resultante se opera de igual manera que en el suelo.

Para que todo esto pueda implementarse de igual manera que en el caso del suelo, a partir de ahora se tratará al cielo como un objeto más que se introduce en la escena, pero tratándolo por separado para que no se le aplique iluminación ni reflexión ni refracción. De esta forma, si el trazador de rayos no encuentra ningún objeto, devolverá el color 0 (negro).

CÓMO ENVOLVER UNA ESFERA

Envolver una esfera no es un proceso tan evidente como los casos anteriores. Una vez más, es necesario echar mano de algunas fórmulas de geometría.

Para empezar, hay que definir dos vectores *normalizados* que sirven para orientar la textura. Uno de ellos apunta hacia un punto de referencia en el ecuador y se llama "Se". El otro apunta hacia el polo norte y se llama "Sp". Con estos vectores se puede dar el efecto de que la pelota gire en diferentes secuencias de una animación. En la figura 3 se pueden apreciar estos dos vectores. Cuando se crea un objeto esfera, los valores por defecto de éstos son, respectivamente, (1, 0, 0) y (0, 1, 0). Posteriormente se pueden rotar o modificar como se quiera, siempre que continúen siendo normalizados y perpendiculares entre sí.

El objetivo es obtener la longitud y la latitud del punto que se está estudiando. Estos valores servirán para obtener U y V. Se empezará por este último.

Las texturas proporcionan un gran realismo

V varía según lo hace el ángulo B. Este ángulo se puede obtener mediante el arco coseno del producto escalar entre los vectores "Sp" y N (vector normal a la superficie en el punto que se está estudiando). De esta forma, se obtiene :

B = acos (- Sp | N)
V = B / Pi

Una vez más, hay que recordar que "|" representa el producto escalar de dos vectores. Como se puede apreciar en la figura 3, B va a variar entre 0 y Pi. Por lo tanto, al dividir por Pi, se consigue que V varíe entre 0 y 1. Para acelerar un poco esto, se puede tener pre-

calculado 1 / Pi y multiplicar en lugar de dividir.

Para calcular U, se usarán fórmulas clásicas del cálculo de longitudes y latitudes. Así :

```
A = acos (( Se ! N ) / sin ( B )) /
( 2 * Pi )
```

En la fórmula anterior, se ha divido por sin (B), lo cual va a suponer una división por 0 en los polos de la esfera. Si ocurre este caso, simplemente se asigna un valor arbitrario para U, por ejemplo, 0. Para acelerar la fórmula anterior, se tendrá precalculado 1 / (2 * Pi).

Finalmente, la variable U se obtiene comparando el producto vectorial de Sp y Se con el vector normal, para obtener el extremo más próximo del vector Se al punto de intersección estudiado :

```
if ((( Sp & Se ) ! N ) > 0 )
  U = A ;
else
  U = 1 - A ;
```

Hay que recordar que "&" representa el producto vectorial de dos vectores.

En el suelo y en el cielo se emplea la repetición de texturas

Para agilizar el proceso, el producto vectorial, Sp & Se, se puede precalcular.

Una vez calculados U y V, se pueden aplicar los mismos conceptos que en el caso del suelo. De esta forma, *escala_u* indicará el número de veces que se usa el mapa a lo largo del ecuador, *escala_v* indicará cuantas veces se usa verticalmente, y *espejo* indicará cómo se produce la repetición de los mapas : invirtiendo uno sí y uno no o todas las veces al derecho.

Para ello, se multiplica U y V por sus correspondientes factores de escala :

```
U = U * escala_u
V = V * escala_v
```

Para aplicar la variable *espejo*, se procede de forma similar que en el caso del suelo :

```
if ( espejo )
{
  U = modf ( U * 2, & i ) ;
  if ( U > 1) U = 2 - U ;
  V = modf ( V * 2, & i ) ;
  if ( V > 1) V = 2 - V ;
}
else
{
  U = modf ( U, & i ) ;
  V = modf ( V, & i ) ;
}
```

CÓMO ENVOLVER UN TRIÁNGULO

Envolver un polígono genérico de n lados no tiene una solución evidente. Generalmente, se suelen utilizar polígonos de 3 o 4 puntos (la mayoría de las veces de 3), ya que polígonos de muchos puntos suelen acarrear problemas al no ser los puntos perfectamente coplanares (al no estar todos en un mismo plano) y, en lugar de eso, se construyen a partir de varios polígonos de 3 puntos.

Es por eso que sólo se van a estudiar las texturas sobre polígonos de 3 y 4 puntos (el de 4 puntos se verá en el próximo capítulo). Para ello, se han creado una clase nueva : la clase *triangulo*, que exteriormente es muy similar a la clase *poligono*, variando sólo su constructor y permitiendo el uso de texturas, cosa que no se puede hacer con la clase poligono. La función envolver de la clase polígono siempre va a devolver los valores (0, 0).

La clase triangulo hace de forma diferente y más optimizada el estudio de la inclusión del punto de intersección en el polígono. En este estudio, además se obtienen los valores de U y V directamente, por lo que la función envolver siempre va a devolver los últimos valores obtenidos. Hay que tener mucha precaución en este punto, ya que sólo se podrán obtener los valores U y V de puntos en los que se acaba de estudiar la intersección del triángulo con el rayo.

Como se puede apreciar en la figura 4, el punto P, el cual ya se ha obtenido de igual manera que en la clase poligono, se puede expresar como la suma de dos vectores, multiplicados por las variables U y V correspondientes :

BIBLIOGRAFÍA

- "An Introduction to Ray Tracing". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Graphics Gems". Editado por Andrew S. Glassner. Editorial Academic Press.
- "Fundamentals of Three-Dimensional Computer Graphics". Alan Watt. Editorial Addison-Wesley


```
VoP = U * VoV1 + V * VoV2
```

Si expresamos esta ecuación usando las componentes de cada punto, se obtiene el siguiente sistema de ecuaciones:

```
xp - x0 = U * ( x1 - x0 ) + V *
( x2 - x0 )
yp - y0 = U * ( y1 - y0 ) + V *
( y2 - y0 )
zp - z0 = U * ( z1 - z0 ) + V *
( z2 - z0 )
```

Este sistema siempre va a tener una única solución. El punto pertenecerá al triángulo siempre que U y V estén comprendidos entre 0 y 1. Para resolver el sistema, se proyecta sobre uno de los tres planos coordenados, tal y como se hace con los polígonos. Si se llama "j" la primera coordenada (que puede ser "x", "y" o "z") y "k" la segunda (ídem), se obtiene el sistema :

```
jp - j0 = U * ( j1 - j0 ) + V * ( j2 - j0
)
kp - k0 = V * ( k1 - k0 ) + V *
( k2 - k0 )
```

Y la solución del problema queda así :

```
U = [( jp - j0 ) * ( k2 - k0 ) - ( kp -
k0 ) * ( j2 - j0 )] / [( j1 - j0 ) * ( k2 -
k0 ) - ( k1 - k0 ) * ( j2 - j0 )]
V = [( jp - j0 ) * ( k1 - k0 ) - ( kp - k0 )
* ( j1 - j0 )] / [( j1 - j0 ) * ( k2 - k0 ) -
( k1 - k0 ) * ( j2 - j0 )]
```

El denominador de ambas divisiones puede ser precalculado. Además, este valor servirá para indicar si el triángulo está bien construido (no es ni una línea ni un punto). Lo que se va a hacer es guardar estos valores y devolverlos en la función envolver.

Una vez obtenidos estos dos valores, se procede como con la esfera para aplicar la escala y el espejo.

INTERRUPCIONES Y MULTITAREA

José María Peco

Seguramente todos los programadores se han preguntado alguna vez cómo el ordenador puede gestionar tantas tareas a un mismo tiempo sin perder el control. Este artículo intenta contestar esta cuestión exponiendo el modo como lo resolvió el D.O.S., el Sistema Operativo para grandes sistemas de IBM antecesor del MVS.

Como ya se comentó en la introducción de los Sistemas Gestores de Bases de datos, se entiende por *multiprogramación* el tratamiento en forma concurrente de más de un programa en tiempo real. Este concepto es el que nos ocupa, pues pretende estudiar cómo un único procesador o CPU puede ejecutar al mismo tiempo, al menos dos programas, el supervisor y un programa de usuario, o programa problema. Hay que tener en cuenta que en un gran sistema, para que se pueda ejecutar un programa en el ordenador, éste debe contar con un programa que controle las operaciones de entrada/salida necesarias para el tratamiento, explotación y puesta a punto de los programas realizados por el usuario o por el constructor. Este programa de control se llama ***supervisor*** y se ubica en la memoria principal, concretamente en las posiciones más bajas, está siempre cargado, y tiene por misión:

- planificar la ejecución de todo tipo de programas
- controlar y coordinar las operaciones de E/S de todos los programas problema.
- Manejar todas las interrupciones.

Programa problema: recibe esta denominación todo programa que contenga operaciones de entrada de datos, proceso de esos datos, y salida de los datos elaborados, y que además se ejecute interactivamente con el supervisor. Se podría decir que los programas problema son los programas escritos por los programadores(nomina, facturación, etc..) , y algunos programas ofrecidos por los constructores (compiladores, sort, etc.). El supervisor se diferencia de estos programas porque no elabora información, sólo controla el sistema.

ESTADOS DE PROGRAMA DE LA CPU

La CPU puede encontrarse en diferentes situaciones o estados a lo largo de un trabajo, existiendo los 8 estados de programa que se muestran en la figura 1.

No todos los estados son excluyentes, por los que la CPU puede estar al mismo tiempo en más de un estado.

Ejemplo: Cuando la CPU se encuentra en estado operativo, de funcionamiento y de problema, puede estar también en estado enmascarable o en estado interrumpible. En cambio, si la CPU se encuentra en estado de parada no puede estar en otro estado pues no puede ejecutar instrucciones.

- La CPU se encuentra en estado de *supervisor* cuando esta ejecutando instrucciones de supervisor.
- La CPU se encuentra en estado de *espera* cuando no se procesan instrucciones pero se mantiene a punto de estar en funcionamiento.
- La CPU se encuentra en estado *operativo* cuando esta o ejecutando instrucciones o en un estado de *interrupción*, es decir, o esta en estado de *funcionamiento* o esta en estado de *espera*.

Los 8 estados de programa se diferencian entre si por los siguientes tres motivos:

- el modo como funciona la CPU: en cada estado de programa, hay ciertas cosas que la CPU puede hacer y otras que no.
- el modo como se indica la situación de la CPU, es decir cómo se sabe que la CPU se encuentra en cierto modo: La forma de indicar el estado de la CPU es bien por medio de alguna luz de la consola o bien por medio del estado de cierta o ciertas posiciones en algún lugar de memoria (PSW).

- El modo como se pasa a dicho estado: El modo de pasar de un estado a otro se hace bien por medio de alguna tecla de consola, o bien por lo que se llama

interrupciones.

Normalmente el sistema estará en estado de *funcionamiento*, ejecutando instrucciones del programa problema o del supervisor. Pero, ocasionalmente ocurrirán operaciones que obliguen a la CPU a pasar un programa al estado de *espera*, es decir a parar la ejecución de sus instrucciones.

INTERRUPCIONES:

Cuando un mensaje llega al ordenador, su llegada debe crear una interrupción en el proceso que se esta ejecutando, de forma tal que se pueda leer el mensaje y almacenarlo hasta que pueda ser tratado.

Realmente una interrupción es una solicitud hecha a la CPU para cambiar el estado del programa que se esta ejecutando. Pero este cambio de estado de funcionamiento a espera, se puede realizar tanto por la aparición de una interrupción como por la ejecución de la instrucción de ensamblador **LOAD PSW.** Por ejemplo, siempre que se deba atender una operación de E/S la CPU deberá estar en estado *Supervisor*, ya que es este programa el que controla estas operaciones. Basta pensar que antes de que apareciese la operación de E/S la CPU estaba ejecutando instrucciones del programa problema y por lo tanto, se encontraba en estado de *problema*; pero, al aparecer la operación de E/S, este programa problema solicita que la CPU pase del estado de *problema* al estado de *supervisor*, con lo cual se habrá ocasionado una interrupción de llamada al supervisor, para poder realizar la operación de E/S.

CLASES DE INTERRUPCIONES:

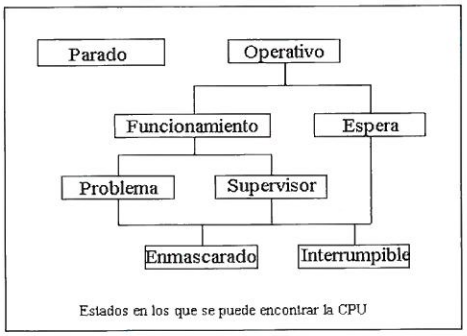
- E/S : se produce cuando termina una operación de E/S bien desde un dispositivo o hacia un dispositivo.
- Externa: se produce cuando el operador aprieta la tecla de interrupción que tiene la consola. En este tipo de interrupción la CPU pasa del estado de espera al estado de funcionamiento. Por ejemplo una señal del contador de tiempos.
- De programa: Se produce cuando un resultado aritmético es demasiado largo para entrar dentro del registro o del área de almacenamiento prevista, o cuando se quiere tratar caracteres alfabéticos como numéricos, etc.
- Por error de maquina: se produce cuando existe una deficiencia de funcionamiento en unidades que no son de E/S. Un ejemplo: pudiera ocurrir que al transferir datos de un sitio a otro, los circuitos ganasen o perdiesen un bit, generándose un error de paridad.
- De llamada al supervisor: esta interrupción se produce cuando el programa problema ejecuta la instrucción de ensamblador **SVC** (SUPERVISOR CALL) de llamada al supervisor. Con esta instrucción se pasa del estado de problema al estado supervisor.

Los programas problema pueden contener muchas condiciones que exijan pasar al estado de supervisor, pues éste debe gobernar todo el sistema, y por tanto, debe recibir el control cuando se quieran leer o escribir datos, cargar nuevos programas a ejecutar, poner a punto el contador de tiempos, cancelar programas que se están ejecutando, etc...

PSW Programa Status Word : Palabra de estado de programa.

Esta es una doble palabra (*) de información, cuyo contenido, mostrado en la figura 2, refleja la situación lógica del ordenador respecto al programa problema que se esta ejecutando, y dispone entre otros, de los campos que reflejan las condiciones lógicas siguientes:

- *Estado* en que se encuentra la CPU
- *Dirección* de la próxima instrucción a ejecutar
- *Relación* (Mayor, menor o igual) entre los últimos datos comparados en el programa (código de condición).



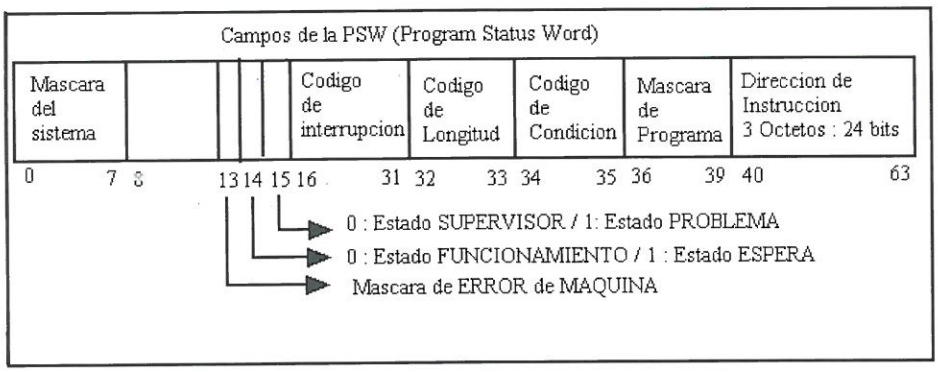
Estados en los que se puede encontrar la CPU

Existen varias PSW, sin embargo en un momento determinado solamente actúa una, recibiendo ésta la denominación de **PSW ACTUAL**. Todas las PSW están almacenadas en lugares concretos y predeterminados de la memoria, formando la PSW actual parte de la circuiteria interna de la maquina.

En la misma figura 2 se puede apreciar cómo el valor de los bits 13, 14 y 15 se relaciona con los estados de la CPU, aunque estos valores tienen sentido sólo en la **PSW ACTUAL**.

Por otra parte, y como ya se ha dicho anteriormente, el programa puede pasar al estado de espera o cuando se produce una interrupción, o cuando se ejecuta la instrucción ***LOAD PSW***. Esta instrucción, que se ejecuta siempre en estado supervisor, escoge una PSW de la memoria, y la transfiere a otras posiciones de memoria, concretamente a las posiciones destinadas a la PSW actual. Así, cuando el programa problema pretende realizar una operación de E/S ejecutará entre otras, la instrucción de ensamblador *SVC*, que provocará la interrupción, con lo que la PSW actual será sustituida por una PSW nueva que tendrá un 0 en el bit 15.

Otro problema con el que se encuentra la CPU es que no siempre puede atender a todo tipo de interrupciones. Así, si


Campos de la PSW (Program Status Word)
Mascara del sistema
Codigo de interrupcion
Codigo de Longitud
Codigo de Condicion
Mascara de Programa
Direccion de Instruccion 3 Octetos : 24 bits
0 7 8 13 14 15 16 31 32 33 34 35 36 39 40 63
0 : Estado SUPERVISOR / 1: Estado PROBLEMA
0 : Estado FUNCIONAMIENTO / 1 : Estado ESPERA
Mascara de ERROR de MAQUINA

el supervisor esta procesando una interrupción de entrada/salida, toda interrupción de E/S que se produzca procedente de los canales(**) quedará enmascarada, es decir, pendiente de ejecución.
El enmascaramiento se logra por la presencia de bits 0 en los campos de máscara del sistema de la PSW actual, dedicándose el bit 7 para las interrupciones externas, y el resto para las de E/S.
Para enmascarar la CPU contra las deficiencias de funcionamiento de la maquina, se incluye el bit 0 en la *posición 13* de la PSW actual, es decir, en el campo de máscara de *error de maquina*.
Por su parte, las interrupciones de programa, provocadas por resultados aritméticos excepcionales y por otras condiciones se enmascaran incluyendo bits 0 en el campo de máscara del programa.
La interrupción de supervisor no tiene máscara, por lo que siempre era atendida.
El paso de 0 a 1 o de 1 a 0 de estos bits, se realiza mediante la ejecución de determinadas instrucciones de ensamblador. Así, para poner ceros o unos en la *máscara de programa*, el programa problema puede ejecutar la instrucción *SET PROGRAMA MASK*. Y, para cambiar los bits del campo *máscara del sistema*, se ejecutará la instrucción *SET SYSTEM MASK*. Esta instrucción sólo se puede ejecutar en estado de supervisor, aunque funciona igual que la *SET PROGRAMA MASK*.

BAILE DE LAS PSW.

Ya se ha comentado que hay movimientos de PSW, lo cual implica que hay varias PSW, pero una única es la que influye sobre el sistema en un momento determinado, es la **PSW ACTUAL**. Estos movimientos de PSW tienen su justificación, ya que se debe preservar el estado lógico del sistema antes de la interrupción para poder restablecerlo después de la interrupción.

Al sustituir la PSW actual por una nueva, aquella no se pierde, sólo se almacena en otra posición concreta de memoria. Pero como una vez almacenada su contenido ha perdido actualidad, pasa a llamarse **PSW anterior**.

Así pues, se pueden apreciar los siguientes términos:
PSW nueva: Es la PSW que, como consecuencia de la interrupción, sustituye y se convierte en PSW actual.
PSW anterior: Es la PSW que fue sustituida y queda almacenada en espera de que se necesite de nuevo.
PSW actual: es la PSW que se usa durante la ejecución de una instrucción. Es la que influye sobre el funcionamiento del sistema.

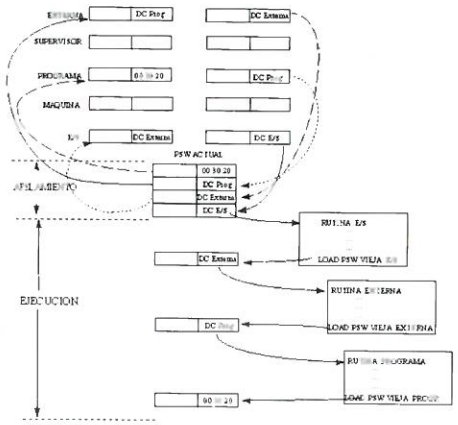
Hay 5 PSW nuevas, una por cada tipo de interrupción, y las posiciones de memoria predeterminadas para contener las distintas PSW son las especificadas en la figura 3.

QUÉ OCURRE CUANDO SE PRODUCE UNA INTERRUPCIÓN:

El concepto clave para entender este entramado de PSWs es que dependiendo de cual haya sido la clase de interrupción, la **PSW actual** se almacenará en la posición de memoria correspondiente a la **PSW anterior** de la interrupción producida; y, la **PSW nueva** que contiene, entre otras cosas, la dirección de comienzo de la rutina encargada de tratar la interrupción producida, será la que se cargue como PSW actual. Es decir, la PSW actual se convierte en PSW anterior, y se almacena en el lugar previsto para la clase de interrupción generada; y la PSW nueva correspondiente a este tipo de interrupción, se convierte en PSW actual.
Es lógico, por tanto, que al haber 5 PSW nuevas, una por cada tipo de interrupción, cada una apunte a la primera instrucción de la rutina del supervisor que se ocupa de esta clase de interrupción.
EJEMPLO: Si se especifica en el programa problema el tratamiento aritmético de un campo alfabético, se producirá un error que generara una interrupción, que por su naturaleza será de *programa*, por lo que habrá que buscar la nueva PSW en la posición 104 de memoria, la cual contiene la *dirección* de la primera instrucción de la rutina del supervisor que maneja las interrupciones de programas.

DIRECCIÓN DE INSTRUCCIÓN:

El campo *dirección de instrucción* contiene la dirección de la próxima instrucción a ejecutar por la CPU. Ahora bien, su contenido depende de que sea PSW nueva, en cuyo caso contiene la dirección de comienzo de la rutina que trata la interrupción, o que se corresponda con la ejecución de un programa, en cuyo caso contiene la dirección de la próxima instrucción del programa problema que se esta ejecutando. En cualquier caso la CPU sólo ejecuta la instrucción a la que apunta la dirección de la PSW ACTUAL.



CÓDIGO DE LONGITUD:

Generalmente las instrucciones se ejecutan en la misma secuencia en la que se encuentran almacenadas. Las instrucciones tienen varias longitudes, de 1 , 2 o 3 medias palabras. Pues bien, uno de los campos de la PSW contienen la longitud, en medias palabras, de la instrucción que se esta interpretando. Este campo se llama *código de longitud de instrucción*.

CÓDIGO DE CONDICIÓN

Por otra parte, en los programas suelen existir puntos donde la secuencia de las instrucciones cambia. A estos cambio de secuencia se les llama bifurcaciones, y pueden ser condicionales e incondicionales.
Las bifurcaciones condicionales son aquellas que están basadas en el resultado de una operación de comparación anterior de datos. En cambio, las bifurcaciones incondicionales se producen siempre que se ejecuta una sentencia de bifurcación, y son independientes de cualquier condición.
También se produce un cambio en la secuencia de instrucciones, cuando se

CORREO LECTORES

Para cualquier duda referente a los temas/artículos tratados en la revista, escriba una carta a:

Sólo Programadores
Referencia: *Correo Lectores*
TOWER COMMUNICATIONS
C/ Marqués de Portugalete, 10 Bajo
Madrid 28027

CÓMO SUSCRIBIRSE A

SÓLO PROGRAMADORES

Revista práctica para usuarios de PC

Suscríbase enviando este cupón por correo o fax (91) 320.60.72, o llamando al teléfono (91) 741.26.62 Horario 9 a 14 y 15:30 a 18:30 h.

6614211 =ERIKA

Deseo suscribirme a la revista SÓLO PROGRAMADORES acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (12 números) **por sólo** 11.950 ptas. (ahorro 20%). ☐ Estudiantes carreras técnicas: 8.950 ptas. (ahorro 40%)

*** ESTA OFERTA ANULA LAS ANTERIORES, DESCUENTOS NO ACUMULABLES.**

Nombre y apellidos.....Domicilio.....

Población.....C.P .....Provincia.....Telf.....Profesión.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº ____ ____ ____ ____

Fecha de caducidad de la tarjeta.....Nombre del titular, si es distinto.....

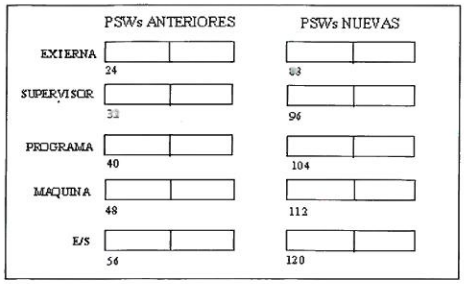
☐ Domiciliación bancaria.
Señor Director del banco .....
Población .....
Ruego a vd. que se sirva cargar en mi ☐ cuenta corriente ☐ libreta de ahorro número.....
el recibo que le será presentado por TOWER COMMUNICATIONS, S.R.L.
como pago de mi suscripción a la revista SÓLO PROGRAMADORES.

ENTIDAD	OFICINA	DC	Nº CUENTA

Firma:

☐ Contra-reembolso del importe más gastos de envío.
☐ Cheque a nombre de TOWER COMMUNICATIONS S.R.L., que adjunto.
☐ Giro Postal (adjunto fotocopia del resguardo).

Rellena este cupón y envíalo a:
TOWER COMMUNICATIONS SRL
C/ Marques de Portugalete 10, Bajo
28027 Madrid.



pasa del estado de problema a estado de supervisor, pues la ejecución de instrucciones pasara a otra dirección diferente ya que ha habido cambio de PSW. Este cambio de secuencia no es propiamente una bifurcacion, pero la información de comparación generada en los pasos anteriores en el caso de una bifurcación condicional, debe guardarse hasta que se interprete la siguiente instrucción del programa que se esta ejecutando. Por esta razón, en la PSW existe un campo que guarda esta información, y que se llama *código de condición.*

Este campo refleja las condiciones existentes después de ejecutar una instrucción. No todas las instrucciones modifican el código de condición de la PSW. Este campo se encuentra en las posiciones 34-35 de la PSW y nos puede indicar :

1.- la relación (mayor, menor o igual) de una cantidad con otra.
2.- si una cantidad es mayor, menor o igual a cero.
3.- si una cantidad es distinta de cero.
4.- si el comienzo de una operación de E/S ha tenido éxito o no .

CONTADOR DE TIEMPOS:

Otra de las funciones del supervisor es la de suministrar servicios de contador de tiempo, para lo cual se coloca en la memoria, concretamente en la posición 80 hexadecimal (128 decimal) cualquier valor de tamaño palabra. Este valor ira disminuyendo automáticamente en base a la señal de sincronismo del sistema, y cuando llegue a 0 generará una interrupción externa. Así mismo, esta función permite al programa problema pedir al supervisor información sobre el contenido del campo que señala el tiempo.

PRIORIDADES DE INTERRUPCIONES:

Para conocer la prioridad con la que se tratan las interrupciones, se debe suponer que se producen al mismo tiempo los cinco tipos de interrupción descritos.

En primer lugar, hay que indicar que esto no es posible, ya que las interrupciones de supervisor y de programa son excluyentes. Esto es así porque para que se produzca una interrupción de supervisor se debe ejecutar la instrucción de programa *SVC*, y si esta instrucción es correcta, no se puede motivar un *PROGRAMA CHECK* (error de programa) pues si se ejecuta la *SVC*, al ser una instrucción de programa, es porque no se ha interrumpido el programa.

Supongamos entonces que se producen simultáneamente 4 interrupciones

- externa
- supervisor o programa (son excluyentes)
- maquina
- E/S

Las peticiones de interrupción se apilan en un orden determinado de antemano:

- máquina
- programa- supervisor
- externa
- E/S

Sin embargo, se atienden de acuerdo con otro orden también predeterminado:

- máquina
- E/S
- Externa
- Programa-supervisor

Como se ve, las interrupciones de error de máquina constituyen un caso especial, y esa razón es la que hace que aparezcan en cabeza de ambas listas. Si llegan las 4 interrupciones a la vez, la de máquina es la primera en apilarse, y la que comienza a ejecutarse, no atendiendo las otras tres. La razón es clara, pues la interrupción de maquina indica que la circuitería del ordenador está fallando, y por lo tanto, toda operación que se realice después no es fiable.

Generalmente, cuando se produce una *interrupción de máquina*, cesa de ejecutarse la instrucción en curso y se graba un tipo especial de información que posteriormente se consultará para su diagnostico y reparación, mientras que cuando se producen los otros tipos de interrupción, se atienden cuando la CPU termina la operación que esta ejecutando, y antes de que comience la siguiente.

Así pues, se reduce la petición simultánea a 3 interrupciones: E/S, P-S, y Externa, y supongamos que la PSW actual, apuntaba a la dirección 003020 en el momento de producirse las tres interrupciones.

La primera, apilada según muestra la figura 4, es la de **programa**, por lo tanto la PSW actual pasa a PSW anterior de programa, y la PSW nueva de programa pasa a ser actual; pero como hay otra interrupción encolada, pasa a tratarse la **externa** pendiente, para lo cual ocurrirá lo mismo que con la de programa, es decir, la PSW actual pasa a PSW anterior externa, y la PSW nueva externa, pasa a actual. Pero entonces se pasa a atender a la de **E/S**, por lo que la PSW actual (que contiene la PSW nueva externa) pasa a PSW anterior de E/S, y la PSW nueva de E/S pasa a actual.

Llegados a este punto, y como no hay más interrupciones pendientes, la CPU pasa a ejecutar las instrucciones de la rutina que trata las **interrupciones de E/S**, pues el campo de dirección de instrucción apunta a la primera instrucción de esta rutina, la cual termina cargando la PSW anterior de E/S, por lo que una vez ejecutada comienza a ejecutarse la rutina que trata **las interrupciones externas**, y así sucesivamente hasta terminar cargando la PSW anterior de interrupciones externas, que contiene la PSW nueva de **interrupción de programa**, que a su vez terminara cargando la PSW anterior de interrupción de programa que contiene la **PSW actual del programa** que se estaba ejecutando en el momento de producirse las interrupciónes, por lo que proseguirá con la ejecución del programa desde donde lo había dejado.

Este baile de PSWs muestra cómo el orden de ejecución de interrupciones resulta ser el inverso al de apilamiento, pero sobre todo muestra el mecanismo diseñado para poder controlar la ejecución de al menos dos programas de forma concurrente, tal y como se había fijado como objetivo de este artículo.

ESTRUCTURA BÁSICA DEL NÚCLEO

David Aparicio

Este mes nos ocuparemos de explorar la organización del núcleo principal de Linux, y sus principales componentes.

Para los lectores interesados en Linux desde un punto de vista pedagógico, el método más rápido para aprender sobre este sistema es estudiar las fuentes del mismo. El núcleo del sistema es la parte de software donde se interacciona más de cerca con todas las características específicas de nuestra máquina. Por ello, es una fuente de información importante para desarrolladores que deseen acceder al hardware del PC, bien para construir *drivers* para otros sistemas operativos, o para mejorar el funcionamiento del propio Linux, y representan la mejor documentación técnica sobre el mismo.

INTRODUCCIÓN

Todos los fuentes del núcleo Linux se encuentran organizados a partir del directorio */usr/src/linux*, y nos referiremos a lo largo de este articulo a rutas relativas a dicha posición.

La estructura a revisar se obtendrá a partir de archivos que encuentran en la serie D de todas las distribuciones Slackware. Nos hemos basado en la versión 1.2.13, la última estable (de explotación) disponible a día de hoy. Una vez instalado, observaremos que el núcleo de Linux se divide en cinco zonas básicas:

- arranque y configuración
- kernel básico
- sistema de ficheros (VFS)
- gestión de memoria
- interfaz con periféricos

A continuación veremos muy por encima los componentes del sistema, según la estructura de directorios en que se organiza. Observe el cuadro general (figura 1).

DETALLES DEPENDIENTES DE ARQUITECTURA: arch/

Desde hace algún tiempo, parte del esfuerzo de Linus Torvalds se ha concentrado en independizar lo más posible el sistema operativo de la arquitectura original en la que se concibió, el 386. La estructura de directorios que cuelgan de *arch/* contienen la parte dependiente de cada arquitectura. Aquí se encuentra el código especifico de cada sistema, bien por estar escrito en ensamblador (por ser una zona crítica), o por dedicarse a controlar singularidades del mismo.

A lo largo del tiempo, se ha reducido este código hasta lo imprescindible, puesto que así se propicia el crecimiento multiplataforma de Linux. Las prestaciones del mismo no han variado apenas, y a cambio se ha logrado una mayor flexibilidad. Actualmente, (en la versión 1.2.13), existe código para los siguientes microprocesadores:

- Intel x86
- Digital ALPHA
- Sun SPARC
- Dec MIPS

La parte más desarrollada, y única hasta hace poco tiempo, es la que tomaremos para nuestro pequeño

repaso. La estructura para la arquitectura más popular del momento, Intel, está contenida en el subdirectorio *arch/i386/*. Sin embargo, y en el momento de escribir ésto, acaba de ser lanzada una distribución para arquitectura Alpha, de Digital, que está destinada a ser la gama alta de estaciones de trabajo Linux, con rendimientos superiores a la arquitectura Intel (incluido Pentium Pro) por un precio similar.

El fichero *config.in* contiene la utilidad de configuración para esta arquitectura concreta, y se utiliza cuando invocamos el comando *'make config'* al reconstruir el núcleo.

El directorio *math-emu/* es específico de esta arquitectura, puesto que ofrece la emulación del x387. Los otros sistemas suelen no disponer de esta parte, puesto que dichos microprocesadores siempre incluyen un *core* de coma flotante. Este es un ejemplo de código dependiente del micro por su naturaleza especifica. En el mismo caso se encuentra el directorio *ibcs/* que cumple funciones de compatibilidad con otros binarios Unix en arquitecturas Intel (formatos ELF/SVr4 y COFF/IBCS).

El directorio *arch/i386/boot/* contiene información sobre la forma en la que se inicia Linux:

- *bootsect.S*: el código 8086 de arranque e inicialización específico para PC. Cuando la BIOS encuentra una unidad de disco, carga el sector de arranque y le pasa el control. Por ello, ocupa exactamente 512 bytes y aquí debe residir el código que carga todo el resto del núcleo.

- *setup.S:* También contiene código general 8086 de inicialización de datos en modo real, obtención de información sobre vídeo y discos mediante la BIOS, y paso a modo protegido 386.

El directorio *arch/i386/compressed/* contiene el código de preparación del núcleo en memoria (descompactación) una vez que el cargador lo ha dejado allí. Linux se carga comprimido para que ocupe menor espacio de disco, pero debe ser descomprimido "sobre la marcha" antes de poder ser ejecutado.

arch/i386/tools/ es un directorio heredado de la estructura original del Minix, y contiene las utilidades que organizan y unen los componentes básicos que integran la imagen del núcleo una vez que se ha compilado. *arch/i386/zImage* es el fichero resultante de compilar el núcleo, y debe ser instalado según el método de arranque empleado (LILO, disco de arranque o LOADLIN).

El directorio *arch/i386/mm/* contiene código que se encarga de tratar excepciones de acceso a memoria, faltas de página, inicializa el sistema de memoria y construye la estructura inicial de remapeo con la que funciona el núcleo.

El directorio *arch/i386/kernel/* esta integrado por el código que inicializa el sistema y cumple funciones de bajo nivel, negociando el arranque con la BIOS PCI si está disponible, tratando traps del micro e interrupciones, y organizando la señalizacion de bajo nivel en general.

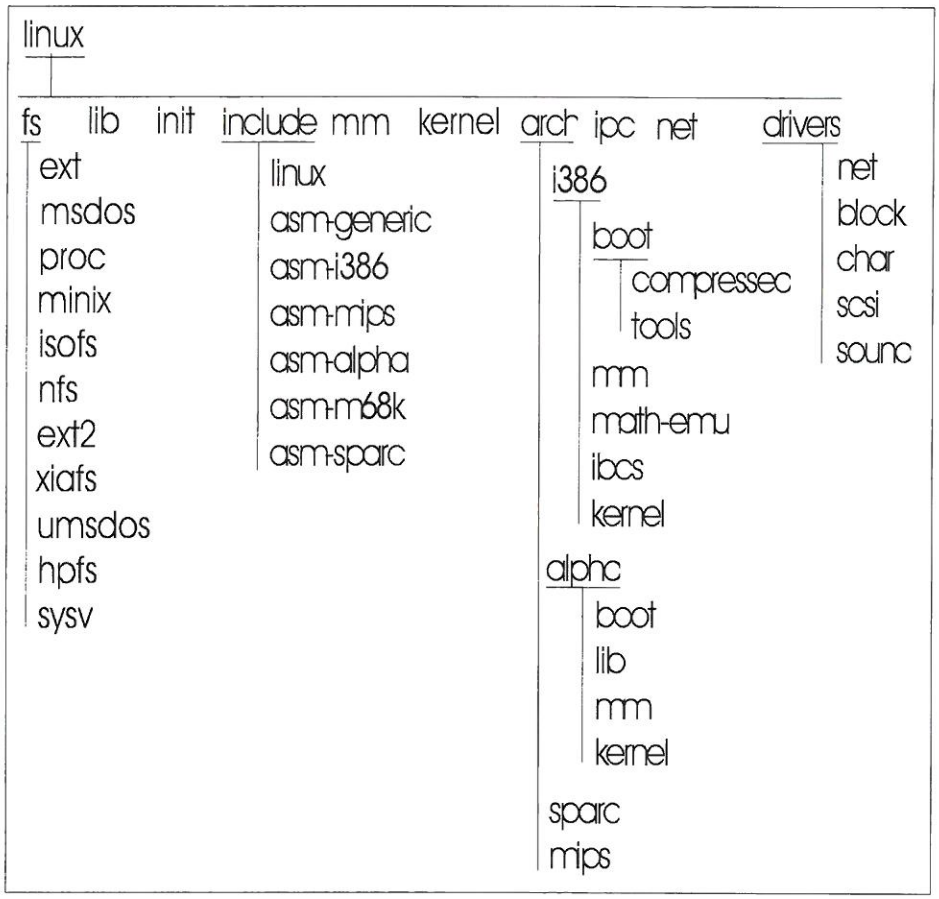


Figura 1: Estructura de directorios.

La estructura de Linux se separa en una parte dependiente de la arquitectura y otra genérica

INICIALIZACIÓN: init/

Cuando el sistema de arranque pasa a modo protegido, se empieza a ejecutar el código de este directorio. Según la configuración que se haya seleccionado al compilar, *main.c* llamará a la rutina de inicialización de cada zona del núcleo. Con ello, se pasa a autodetectar el hardware, y se registran puertos, IRQ, líneas de DMA, y cualquier aspecto de cada dispositivo.

También se analizan y pasan a cada manejador los parámetros de arranque especificados por el usuario.

CONSTANTES Y TIPOS: include/

En esta estructura se localizan tanto los ficheros de declaraciones comunes para todas las partes del núcleo, como el interfaz con los programas de usuario.

El directorio *include/asm/* (correspondiente a alguna de las arquitecturas disponibles) contiene declaraciones para los ficheros en ensamblador, mientras que *include/linux/* se empleará para los ficheros de C.

Por ejemplo, el fichero donde se declaran los números que identifican a cada tipo de dispositivo ("número mayor de dispositivo") lo podremos encontrar en *include/linux/major.h*

PUNTOS DE ENTRADA: lib/

Cuando realizamos un programa, no se necesita "reinventar la rueda", sino que se utilizan servicios del sistema operativo para realizar muchas funciones de tipo genérico, en base a lo que ofrezcan las librerías de usuario. Un grupo de dichos servicios se resuelven en la propia librería, sin intervención del núcleo (por ejemplo, el tratamiento de cadenas de caracteres), mientras que otros, (por ejemplo, la apertura de un fichero), acaban desencadenando una comunicación entre el lado de usuario que solicita el servicio y el lado del núcleo que lo proporciona. Este directorio contiene el código que trata dichas solicitud en el lado del núcleo y el pase de parámetros correspondiente.

Todos los servicios se basan en la interrupción 80h (en el caso de la arquitectura Intel), y son utilizados por más de un centenar de funciones.

NÚCLEO: kernel/ ipc/ y net/

En estos directorios encontramos la personalidad de Linux, el comportamiento del núcleo básico. Este código es independiente de plataforma, y trata de aspectos de creación y terminación de procesos, tratamiento de alto nivel de interrupciones, mensajes informativos del núcleo, asignación de recursos, planificación y temporizaciones.

Aquí, comentaremos un detalle sobre las interrupciones de Linux. Cuando un dispositivo quiere informar de la llegada de un evento al manejador, produce una interrupción. En la arquitectura Intel, esto llama a un vector cuyo código se activa para atender a dicho evento. Durante este periodo, se bloquea la atención a nuevas interrupciones, por lo que el tratamiento debe ser rápido y eficiente. Este tratamiento se realiza en *arch/i386/kernel*. Debido al escaso tiempo disponible, Linux anota el evento ocurrido junto a los datos recogidos, y deja que sean tratados posteriormente por otro manejador de mayor nivel, y que sí podra ser interrumpido. Esta segundo evento (software) se trata en el nivel del núcleo que estamos tratando en este apartado.

El subdirectorio *ipc/* se incluye opcionalmente en el núcleo, y proporciona servicios de señalización para comunicación interprocesos al estilo de Unix System V (semáforos y mensajes). El emulador DOS, por ejemplo, utiliza estos servicios.

El sistema de red también es opcional, y trata del aspecto general de comunicaciones. En el núcleo de explotación (1.2.13) sólo existe código para TCP/IP, pero en el núcleo de desarrollo actual (1.3.xx) se está incorporando AppleTalk, Novell/IPX y otros protocolos.

Está en desarrollo el soporte de redes NetWare y NT como servidor de recursos

SUBSISTEMAS (GESTORES): fs/ y mm/

En la estructura heredada de su origen (Minix), Linux subdivide los servicios que ofrece en dos partes. Por un lado, los servicios del sistema de ficheros se implementan en el subdirectorio *fs/*, y se denomina, genéricamente, VFS (Virtual File System).

En Linux, cada tipo de sistema de ficheros se incorpora a la estructura general mediante la primitiva *mount*. Cuando montamos un nuevo disco, se identifica el tipo de estructura y se mapean las llamadas asocia al manejador de cada sistema en particular. A partir de ese momento, las peticiones de creación/lectura/escritura solicitadas por los procesos para un directorio en particular, llegan al VFS, y desde allí se delegan en el manejador que corresponda al sistema de ficheros involucrado, en base al mapeo mencionado.

Linux se distingue por la variedad de dispositivos que reconoce

Existe un subdirectorio por cada tipo de sistema soportado en el núcleo. Actualmente se entienden el sistema nativo, además del de Minix, OS/2, MSDOS, NFS, ISO(CD), y otros. En las versiones de desarrollo, se incorporan soporte para sistemas WinNT y Novell (NetWare).

El directorio *mm/*, o gestor de memoria, se dedica a coordinar el acceso al mapa de memoria de cada proceso, *swap* a disco y gestión de memoria dinámica. Por un lado, cada proceso dispone de una espacio virtual de 3 Gb (en comparación a los 512 Mb de OS/2 y a los 2 Gb de NT). Dado que un PC normal no tiene tanta memoria física disponible, el sistema operativo se encarga de utilizar una zona de disco como almacenamiento temporal para sacar el código inactivo de la memoria principal (swap). Por otro lado, cada proceso tiene un rango de memoria lineal que comienza en cero, y sigue hasta el máximo disponible. Por ello, existe un remapeo que convierte dichas direcciones lógicas a direcciones físicas en memoria (MMU), todo ello realizado entre este subsistema y el código en *arch/i386/mm*.

MANEJADORES DE DISPOSITIVOS: drivers/

Un sistema operativo no resulta práctico si no puede emplear el hardware periférico de nuestro PC para interaccionar con nosotros. Por ejemplo, OS2 es un sistema operativo muy robusto y versátil, pero el soporte variado de dispositivos es simplemente horroroso. Esta situación ha permitido a otro sistema operativo, a lo largo de los años, superarle en ventas y popularidad, a pesar de su pobre calidad técnica, gracias a facetas tales como un soporte de dispositivos adecuado.

Si una gran empresa no ha conseguido involucrar al mercado en la creación de *drivers* para su sistema, Linux, que no goza de ningún apoyo comercial y no tiene una coordinación centralizada, debería tener todavía más problemas. Sorprendentemente, en este apartado es donde Linux tiene un mayor potencial. Los mismos usuarios que disponen de hardware novedoso pueden contribuir al crecimiento del sistema, estudiando las fuentes del sistema, y observando el comportamiento de los controladores DOS para dicho producto. Así, a finales de 1995 por ejemplo, está en fase de pruebas el controlador de unidades Zip (discos removibles de 100Mb de bajo coste).

Si analizamos la estructura de directorios más voluminosa de Linux, observaremos que las áreas en las que se dividen los manejadores para Linux son:

- dispositivos de caracteres: *char/*
- dispositivos de bloque: *block/*
- dispositivos SCSI: *scsi/*
- dispositivos de red: *net/*
- dispositivos de sonido: *sound/*

Los periféricos de caracteres son aquellos que trabajan con accesos secuenciales en grupos variables de caracteres, como impresoras, líneas serie, pantallas, teclados y terminales.

Los periféricos de caracteres son aquellos que trabajan con accesos secuenciales en grupos variables de caracteres

Los periféricos de bloque se caracterizan por accesos aleatorios en grupos de caracteres de tamaño constante, y engloba discos duros, discos ram y lectores cdrom que no sean de tipo SCSI. Estos últimos tienen su propio subdirectorio, donde se tratan las tarjetas soportadas que controlan este bus.

La controladora PCI SCSI más popular, la 2940 de Adaptec, no tiene soporte oficial en la versión 1.2.13 de Linux, pero existen un parche para actualizarse. Quizá en los próximos meses aparezca la version 1.2.14 para incluir este driver concreto. Los usuarios interesados deberán pasar a versiones del Linux 1.3.xx si desean utilizar su controladora.

Las principales áreas en las que el lector puede experimentar son los controladores de dispositivos

El subdirectorio de sonido ya se trató extensivamente en un anterior artículo, y trata una gran variedad de tarjetas de audio.

La versión del paquete de sonido para Linux, denominado VoxWare, es la 2.9 para la versión 1.2.13 de Linux. Para utilizar algunos programas relativamente nuevos, puede ser necesario actualizarse a la versión 3.0 de VoxWare, disponible en las versiones 1.3.xx del núcleo.

Los usuarios que deseen realizar actualizaciones, pueden obtener los ficheros del núcleo 1.3 incluido en el CD-ROM de este mes, que incluye la version SlackWare 3.0.

CONCLUSIONES

El sistema operativo Linux tiene como aspecto más diferenciador, la disponibilidad de las fuentes sobre todos los aspectos del mismo, para consulta de cualquier usuario. Añadiendo a esto el hecho de que las prestaciones que ofrece son muy avanzadas, nos encontramos ante un sistema que invita al lector a echar un vistazo más a fondo.

En este capítulo hemos intentado enumerar la organización general de los fuentes del núcleo. En próximos números concretaremos ciertos aspectos del mismo. Las principales áreas en las que el lector puede experimentar son los controladores de dispositivos, pero esto no resta que podamos analizar aspectos del mismo núcleo o del sistema de ficheros. Dado que los fuentes de Linux están bastante bien comentados, no deben asustar a ningún desarrollador que tenga conocimientos mínimos en C. Y además nosotros estaremos aquí para ayudarles a comprender Linux. El autor de este artículo está disponible, bien por correo normal, dirigido a la redacción de la revista, o bien por correo electrónico, a la dirección daparic@colibri.tid.es. En el próximo número, comentaremos todos los detalles del nuevo SlackWare 3.0 de forma más profusa que en la sección de instalación del CD de este mes, donde hemos realizado una pequeña introducción. Ánimo y hasta otra.

EL CONTROL DE LOS ERRORES

Juan Manuel y Luis Martín

Algo que puede echar abajo una gran aplicación es la posibilidad de que el usuario, por error, pulse una tecla no deseada, provocando que el programa se quede colgado.

Para dotar a una aplicación de un acabado profesional, es indispensable que ésta disponga de un sistema para el tratamiento de errores. La filosofía con la que el programador debe enfrentarse al diseño del control de los posibles errores es la siguiente: "Si el usuario puede realizar una operación no deseada que provoque un error en tiempo de ejecución, lo hará". Este tipo de errores debe controlarse muy especialmente ya que, además de ,dar un aspecto bastante "chapucero" al programa, puede provocar una pérdida de información vital. Esto se debe a que, generalmente, los programas trabajan con algún fichero abierto, y la información de éste puede que no encuentre actualizada en el momento del fallo.

TIPOS DE ERRORES

A la hora de la elaborar una aplicación en cualquier lenguaje, el programador se encontrará con que aparecen diferentes tipos de errores en las diferentes fases de su creación.

Inicialmente, los errores que pueden considerarse como los menos importantes son aquellos que suceden en tiempo de diseño. Estos errores están causados generalmente por fallos a la hora de teclear el código. Además, será el propio compilador o enlazador el que detecte este tipo de errores, comunicándolo al programador.

Otro tipo de errores son los producidos en la lógica del programa. Un claro ejemplo es aquel en el que, en vez de sumar dos cantidades, se restan. Cualquiera de las dos operaciones es válida para el compilador y para el enlazador, pero los resultados obtenidos por el programa serán erróneos. La única forma de detectar este tipo de errores es mediante la realización de todo tipo de pruebas con el programa, y la posterior comprobación de los resultados obtenidos.

Especialmente importante es la comprobación del programa con datos extremos. Este tipo de datos son los que pondrán al programa al límite del error, y donde suelen surgir los problemas con mayor frecuencia.

Por último están los errores en tiempo de ejecución. Este tipo de errores vienen causados generalmente por una falta de previsión del programador. Casos típicos de este tipo de errores es intentar enviar datos a la impresora cuando ésta se encuentra desconectada, o intentar abrir un fichero que no existe. Este tipo de errores se conoce en programación como *excepciones*, y el presente artículo va encaminado a la prevención de ellos.

Estos errores son los más difíciles de detectar por parte del programador, pero, a su vez, son los más fácilmente detectables por parte del usuario. Siguiendo con el ejemplo de la impresora, cuando el programador va a realizar una prueba de la obtención de los listados, generalmente se asegura de que la impresora esté conectada. Desde el punto de vista del usuario, esta operación no resulta tan lógica. De ahí la importancia de una buena anticipación por parte del programador a los posibles descuidos y equivocaciones del usuario.

EL SISTEMA POR DEFECTO

Por defecto, Clipper dota a todos los programas de un sistema de aviso de

Posiblemente, uno de los aspectos que más van a influir en el éxito de un programa es el control que éste realice sobre cualquier suceso extraño que ocurra durante la ejecución del mismo.

errores. Cuando durante la ejecución de un programa se produce un error, este sistema toma el control. Sin embargo no realiza ningún de tratamiento de errores, sino que se limita a sacar un mensaje por pantalla indicando el motivo del mismo.

Además, en función de la gravedad del error, se ofrece al usuario la posibilidad de reintentar la operación que ha causado el error, o de abandonar la ejecución del programa. Si el error ha sido demasiado grave, el sistema de control de errores únicamente ofrece la posibilidad de abandonar la ejecución.

Los programas realizados con Clipper poseen diferentes subsistemas encargados de realizar las distintas operaciones dentro del programa. Así, cuando se realiza una operación aritmética, Clipper pasa el control al subsistema correspondiente, con el fin de que realice dicha operación. Este subsistema es distinto del encargado de realizar las operaciones con bases de datos, o del encargado de las operaciones con el terminal. Los subsistemas de Clipper pueden consultarse en la tabla 1.

Subsistemas de Clipper

Subsistema	Descripción
BASE	Subsistema de operadores y funciones
TERM	Subsistema para las operaciones con el terminal o consola
DBFNTX	Subsistema para las operaciones de archivos con el driver DBFNTX
DBFNDX	Subsistema para las operaciones de archivos con el driver DBFNDX
DBCMD	Subsistema para las operaciones de archivos con otros drivers
ERROR	Subsistema para controlar los errores producidos al inicializar el objeto Error
TBROWSE	Subsistema para controlar los errores producidos por los métodos de un objeto Browse
FRMLBL	Subsistema para controlar los errores producidos en la generación de informes o etiquetas

Tabla 1.

De esta forma, la detección de los errores se realiza desde los distintos subsistemas de Clipper. Los pasos que se siguen cuando un subsistema detecta un error durante la ejecución de una operación son los siguientes:

1.- El subsistema que detecta el error ejecuta el bloque de código que se encuentra definido para estos casos.

2.- El bloque de código crea un objeto de la clase *Error*, y lo pasa al subsistema que detecto el error.

3.- El subsistema que detectó el error carga los valores correspondientes al error ocurrido en los atributos del objeto y devuelve el control al bloque de código.

4.- Con la información almacenada en el objeto se saca el mensaje por pantalla y, en función del nivel de gravedad, se presentan las correspondientes opciones relativas a las posibles acciones a realizar.

Esta forma de actuar de los programas es siempre la misma. Sin embargo, el programador puede variar las operaciones que se llevan a cabo en el punto cuatro, con el fin de realizar otras operaciones que estime oportunas. Antes de ver la forma de alterar este modo de funcionamiento es necesario conocer algo más sobre los objetos de la clase *Error*.

LA CLASE ERROR

Clipper incorpora una clase, denominada *Error*, para la creación de objetos específicos en los que almacenar la información referente a un error que se haya producido.

Lo primero que debe destacarse es que se trata de una clase atípica, pues no posee ningún método. Los objetos de esta clase únicamente cuentan con atributos, en los cuales se almacenará la información relativa al error que se ha producido.

Los procesos de creación del objeto y de asignación de valores a sus atributos son completamente automáticos, y se encarga de realizarlos el propio Clipper. No es posible obviar este proceso, ni tampoco realizarlo expresamente.

Los objetos de esta clase únicamente cuentan con atributos

La mayor utilidad de este tipo de objetos es la gran cantidad de información que proporcionan sobre el error que se ha producido, dotando así al programador de suficientes datos como para actuar en consecuencia. Los atributos de esta clase pueden consultarse en la tabla 2.

El tratamiento de los errores en Clipper gira entorno al manejo objeto *Error*. Para ello, en el archivo *ERRORSYS.PRG* se encuentran disponibles una serie de funciones, mandatos y estructuras de control que permiten personalizar la forma de actuación cuando se produce un error en tiempo de ejecución.

LA ESTRUCTURA SEQUENCE

La forma tradicional de realizar el tratamiento de errores en las versiones anteriores a la 5.2 de Clipper se basaba en una estructura de control denominada *SEQUENCE*.

```
BEGIN SEQUENCE
  [sentencias]
[BREAK [Expresión]]
  [sentencias]
[RECOVER [USING oError]]
  [sentencias]
END [SEQUENCE]
```

Este tipo de estructura permite ejecutar normalmente una serie de sentencias, hasta que se produce un error. Entonces, el control pasa al sistema de tratamiento de errores, que se encarga de la creación del objeto *Error* correspondiente. Posteriormente, el

Atributos de la clase Error

Atributo	Descripción
Args	Matriz en la que se almacenan los argumentos de la función o el operador que provocó el error
CanDefault	Valor lógico que indica si el subsistema es capaz de subsanar el error
CanRetry	Valor lógico que indica si el subsistema es capaz de repetir la operación
CanSubstitute	Valor lógico que indica si el valor de la operación que ha causado el error puede ser sustituido por otro
Cargo	Variable de propósito general
Description	Cadena de caracteres que contiene una descripción del error
FileName	Nombre del archivo que produjo el error
GenCode	Valor numérico que indica el código de error genérico de Clipper
Operation	Descripción de la operación que se estaba realizando cuando se produjo el error
OsCode	Valor numérico que indica el código de error del sistema operativo
Severity	Valor numérico que indica la severidad o gravedad del error. Sus valores pueden consultarse en la tabla 3
SubCode	Valor numérico que indica el código de error específico del subsistema
SubSystem	Contiene el nombre del subsistema que produjo el error
Tries	Valor numérico que indica el número de veces que se ha repetido la operación que produjo el error

Tabla 2.

control vuelve a la estructura, continuando la ejecución en un grupo de sentencias distinto.

Las sentencias que se ejecutan normalmente son las comprendidas entre *BEGIN SEQUENCE* y *RECOVER*. Las sentencias que se ejecutan cuando se produce un error son las comprendidas entre *RECOVER* y *END SEQUENCE*. A estas sentencias se las denomina *Sentencias de Recuperación*. Además, es posible pasar a estas últimas un parámetro que contiene el objeto *Error* correspondiente. Para ello, debe incluirse la cláusula *USING* en la sentencia *RECOVER*.

En el primer grupo de sentencias de la estructura, es posible generar un error de forma voluntaria. Para ello, debe utilizarse la sentencia *BREAK*. Esta sentencia provoca el paso del control a las sentencias de recuperación de la estructura. Además, puede ir acompañada de un argumento de cualquier tipo de datos que será recibido en la cláusula *USING* de la sentencia *RECOVER*. No es necesario que la sentencia *BREAK* se encuentre físicamente dentro de la estructura *SEQUENCE*. Bastará con que se encuentre en alguna de las funciones o procedimientos invocados por la estructura.

Las estructuras *SEQUENCE* pueden anidarse unas dentro de otras. De esta forma, el control pasará a las sentencias de recuperación de la estructura en la que se haya producido el error o la sentencia *BREAK*.

Si durante la ejecución de las sentencias de una estructura SEQUENCE no se produce ningún error o sentencia BREAK, las sentencias de recuperación no se ejecutarán, pasando el control a la siguiente sentencia a *END SEQUENCE*.

Las sentencias de recuperación deben estar encaminadas a realizar tres tipos de acciones posibles:

- Solucionar el error, si ello es posible.
- Continuar la ejecución, ignorando el error.
- Realizar una salida controlada del programa.

Valores de la gravedad de un error

Valor	Identificador	Descripción
0	ES_WHOCARES	El error es simplemente informativo
1	ES_WARNING	El error es leve, pero puede agravarse
2	ES_ERROR	El error no permite realizar ninguna operación
3	ES_CATASTROPHIC	El error es muy grave, debe abandonarse la aplicación

Tabla 3.

El ejemplo del Cuadro 1 ilustra el manejo de la estructura SEQUENCE. Se trata de un fragmento de programa que salva el error de ejecución producido cuando la impresora no esta preparada.

EL BLOQUE DE CÓDIGO PARA EL TRATAMIENTO DE ERRORES

Cuando se ha producido un error, el sistema siempre genera el objeto *Error* correspondiente. Sin embargo, si el error no se ha producido dentro de una estructura SEQUENCE, el tratamiento es distinto. En este caso, se evalúa un bloque de código especial para el tratamiento de errores. Por defecto, Clipper define este bloque de código en el archivo ERRORSYS.PRG. Para ello, se utiliza la función ERRORBLOCK(), que recibe como argumento el bloque de código que se desea establecer para el tratamiento de errores. Además, esta función devuelve el bloque de código que se encontraba anteriormente establecido.

Cuando se ha producido un error, el sistema siempre genera el objeto *Error* correspondiente

CUADRO 1

```
WHILE .T.
  BEGIN SEQUENCE
    ...
    [Operaciones de impresora]
    ...
  RECOVER USING oError
    IF oError:Severity < 2
       ACCEPT "Error de impresora, ¿Cancelar la
impresión?";
       TO cResp
      IF cResp $ "Nn"
        LOOP
      ENDIF
    ENDIF
  END SEQUENCE
ENDDO
```

Ejemplo de utilización de la estructura SEQUENCE.

CUADRO 2

```
nNum := 10
cCad := "Hola"
...
bAntBloque := ERRORBLOCK( { | oError |
Funcion1( oError ) } )
QOUT( nNum + cCad )          //Error tratado por
Funcion1
bAntBloque := ERRORBLOCK( { | oError |
Funcion2( oError ) } )
QOUT( nNum + cCad )          //Error tratado por
Funcion2
ERRORBLOCK( bAntBloque )
QOUT( nNum + cCad )          //Error tratado por
el sistema por defecto
```

Alteración del bloque de código por defecto.

En el archivo ERRORSYS.PRG existe un procedimiento, denominado ErrorSys(), que establece el bloque de código por defecto para el tratamiento de errores con la siguiente sentencia:

ERRORBLOCK({ | e | DefError(e) })

Se trata de un bloque de código que se limita a invocar la función DefError(), pasándole como argumento el objeto *Error* que se ha creado. Por su parte, la función DefError() se encarga de examinar los atributos del objeto *Error*, visualizando el correspondiente mensaje de error en función de estos.

Como siempre, es posible alterar el contenido de los archivos fuente del sistema para personalizar su funcionamiento. Sin embargo, esto no es recomendable, ya que puede ocasionar conflictos en la elaboración de programas posteriores. Por tanto, la mejor forma de personalizar el tratamiento de errores en un programa consiste en cambiar el bloque de código por defecto, y hacer que éste invoque a funciones definidas por el usuario.

El cuadro 2 muestra un fragmento de código en el que se altera el bloque de código por defecto, permitiendo tratar el error producido al sumar una cadena con un valor numérico de distintas formas.

OTRAS FUNCIONES DE ERRORES

Además de las funciones vistas anteriormente, Clipper dispone de otra serie de funciones relacionadas con el tratamiento de errores. Estas funciones suministran al programador una información adicional sobre el error que se ha producido.

Clipper dispone de otra serie de funciones relacionadas con el tratamiento de errores

ERRORLEVEL([nCodError])

Esta función permite establecer el código de salida que será devuelto al sistema operativo al finalizar la ejecución del programa. Además, devuelve el código de salida que se encontraba anteriormente establecido.

Los códigos de salida son valores que se devuelven al DOS con el fin de que éste tenga conocimiento del estado de terminación de un programa. Estos valores se conocen como *errorlevel* y permiten a los archivos .BAT tomar decisiones para realizar bifurcaciones en sus comandos. Un valor 0 indica que el programa ha finalizado normalmente, mientras que un valor superior indica un determinado tipo de error.

DOSERROR([nCodError])

Cuando se ha producido un error provocado por el sistema operativo, esta función devuelve el código de error del DOS. Sin embargo, la utilidad de esta función está muy limitada, ya que dicho código se encuentra disponible en el atributo OsCode del objeto *Error*.

OUTERR(ListaExpresiones)

Esta función permite dirigir una lista de valores al dispositivo de salida de errores, en vez de al dispositivo de salida estándar. Su funcionamiento es muy similar al de la función OUTSTD(), y permite separar las salidas de mensajes de error del programa de las salidas normales.

NETERROR([lValor])

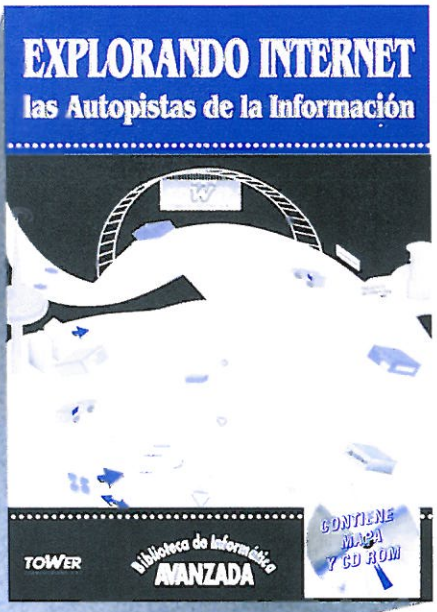
Esta función devuelve .T. cuando se ha producido un error al manejar bases de datos en un entorno de red. Generalmente, los errores de este tipo están relacionados con el sistema de bloqueos para la compartición de archivos. Su utilización ya fue explicada en el artículo sobre programación en red.

CONCLUSIÓN

A la hora de crear una aplicación con Clipper, el programador debe tener presente siempre los posibles errores que puedan producirse en tiempo de ejecución. Mediante las herramientas descritas a lo largo del artículo, el programador puede anticiparse a los posibles errores provocados por el sistema o el usuario, dotando a sus aplicaciones de un aspecto mucho más robusto y profesional.

LOS LIBROS DE INFORMÁTICA MÁS ACTUALES AL MEJOR PRECIO

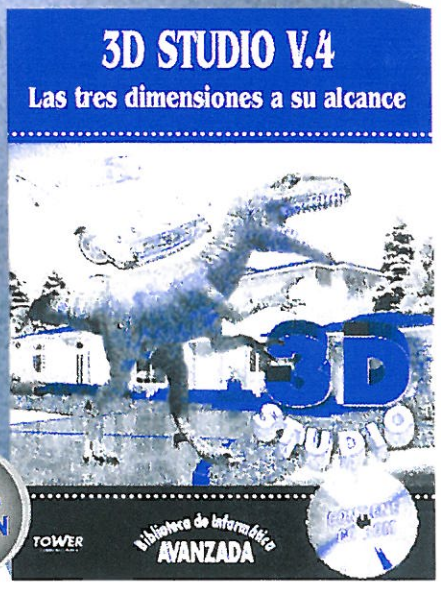
Contiene CD-ROM con una selección de ficheros de sonido, vídeo e imágenes, así como las utilidades disponibles actualmente para Windows 95.



Contiene CD-ROM con 800 programas de acceso rápido a INTERNET

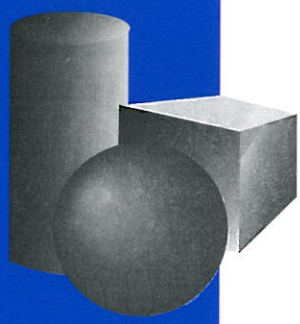
CADA LIBRO POR SÓLO:

1.995 ptas.
IVA INCLUIDO



TOWER COMMUNICATIONS

Solicite catálogo al Tel. (91) 741 26 62



JUEGOS CONVERSACIONALES

Ignacio Cea

Ha llegado ya la hora de que empecemos a integrar todos los conocimientos que se han expuesto a lo largo de las anteriores entregas dentro de clases C++. En el próximo artículo desarrollaremos las bases para la interpretación y ejecución de las órdenes tecleadas por los jugadores.

Es conveniente, desde nuestro punto de vista, que antes de pasar a detallar e implementar las clases encargadas dentro de nuestro framework final de la interpretación y ejecución de órdenes de jugadores, realicemos un breve repaso de lo visto hasta ahora.

En los dos primeros capítulos de la serie vimos, eso si muy brevemente, maneras que podrían ser empleadas por un programador cualquiera para expresar de forma ordenada todas sus necesidades. Los artículos trataban de inculcar la idea de que esas notaciones son útiles tanto para clarificar ideas y tomar visión de conjunto como para comunicarse con otras personas no iniciadas en orientación a objetos.

El siguiente capítulo resaltaba la idea de que dentro del proceso de desarrollo de una aplicación cualquiera no basta con saber quienes son las entidades que participan en el problema y como se relacionan entre sí, sino que también es necesario implementar un solución bajo un ordenador y esto lleva a la aparición de otras clases con sentido estrictamente informático. Se dice que el primer conjunto de objetos pertenece a la fase de análisis y estos últimos a la de diseño.

OBJETIVO

A partir de este momento vamos a tratar de fundir ambas cosas, en lo que ya serán realmente las clases finales del framework para desarrollo de juegos conversacionales.

Parece lógico que la primera parte de la que nos ocupemos sea la interpretación y ejecución de las órdenes de un jugador; es decir que realmente pase algo cuando, por ejemplo, alguien escribe "Coger piedra".

LA BASE INFORMÁTICA DEL JUEGO

Si uno se para a pensar cual de todas las cosas que se manejan dentro de un juego conversacional es la que tiene mayor peso o importancia llegareis, casi con total seguridad, a la conclusión de que son las cadenas de caracteres.

Las cadenas de caracteres son la vía de comunicación entre los distintos participantes del juego y la máquina. Es más cuanto más amplio sea el abanico palabras, descripciones, etc... empleadas dentro del juego mayor será la sensación de realismo del mismo.

Esto lleva a la conclusión de que dentro de nuestro framework debe tener una especial importancia el manejo y tratamiento de cadenas de caracteres.

LA CLASE STRING

Seguro que son muchos los programadores de C que han echado en falta la existencia de un tipo básico semejante al String de otros lenguajes como, por ejemplo, el Pascal. ¿Cuantas horas se habrán perdido empleando instrucciones como "strcpy" o "strcmp". para el manejo de expresiones.

Intentar usar estas órdenes para un sistema donde su necesidad es masiva es una auténtica locura sobre todo si se dispone de un lenguaje como C++ y del concepto de clase. Desarrollemos nosotros la clase String dándole la funcionalidad de un tipo básico cualquiera.

Dentro de las librerías de clases que vienen con la mayor parte de las herramientas de desarrollo actuales (Visual Age C++ de IBM, BorlandC++, etc...) existe ya una clase como el String. Sin embargo coincidirá con nosotros en que con motivos puramente formativos no está de más desarrollar, una vez más, una clase que sustituya al engorroso "chau *".

La clase String

```
//=====================================
=====================================//
// CLASE     : String
                                        //
//  DESCRIPCION:  Cadenas  de  caracteres
//
// AUTOR     : Ignacio Cea Forniés
                                        //
//=====================================
=====================================//

          #if !defined (__STRING_HH__)
          #define __STRING_HH__

          #include <fstream.h>              // Por
canales de comunicación
          extern "C" {
            #include <string.h>             // Por
strcmp..
          };

//————————Características  de  la
clase————————————————//
class String {
  public :
    // Constructores
    String ();
    String (char *,int);
    String (unsigned, const char = ' ');
    String (const char *);
    String (const String &);
    String (const char);
    String &operator = (const String &);
    ~String ();

    // Operadores de cast
    char operator [] (unsigned) const;
    operator char * () const;

    // Operadores de comparación
    int operator ! (void) const;
    int operator == (const String &) const;
    int operator == (const char *) const;
   friend int operator == (const char *,const String &);
    int operator != (const String &) const;
    int operator != (const char *) const;
   friend int operator != (const char *,const String &);
    int operator < (const String &) const;
    int operator < (const char *) const;
   friend int operator < (const char *,const String &);
    int operator > (const String &) const;
    int operator > (const char *) const;
   friend int operator > (const char *,const String &);
    int operator >= (const String &) const;
    int operator >= (const char *) const;
   friend int operator >= (const char *,const String &);
    int operator <= (const String &) const;
    int operator <= (const char *s) const;
   friend int operator <= (const char *,const String &);

    // Operadores matemáticos
    String operator ~ (void) const;
    String operator + (const String &) const;
    String operator + (const char *) const;
friend String operator + (const char *,
const String &);
    String &operator += (const String &);
    String operator - (const String &) const;
    String operator - (const char *) const;
friend String operator - (const char *,const String &);
    String &operator -= (const String &);
    String operator * (unsigned) const;
   friend String operator * (unsigned,const String &);
    String &operator *= (unsigned);
    friend ostream &operator << (ostream &,
const String &);
    friend istream &operator >> (istream &,String &);
    // Manejo friend de los objetos de la
clase
```

↪

```
    // ...estos métodos son friend por intuitividad con
otros lenguajes
    friend unsigned len (const String &s);
    friend String left (const String &,unsigned);
    friend String right (const String &,unsigned);
    friend String mid (const String &,
unsigned pos,unsigned chrs);
    friend String upper (const String &);
    friend String lower (const String &);
    friend String trim (const String &);
    friend int find (const String &,const String &);

  protected :
char *Str;  // String a manipular
unsigned Longitud; // Longitud actual de la cadena
};
//————————————————————————————————
————————————//

    // Métodos inline
    inline String::String () : Longitud (0), Str (NULL) {
/* Vacio */ }
    inline String::String (char *s,int l) : Longitud (l),
Str (s) { /* Vacio */ }
    inline String::~String () { delete Str; }
   inline char String::operator [] (unsigned pos) const
{ return ((pos > Longitud) ? '\0' : this -> Str [pos]); }
 inline String::operator char * () const { return (Str); }
     inline int String::operator ! (void) const { return
((Str) ? 0 : 1); }
 inline int String::operator == (const String &s) const
                          { return (!strcmp (Str,s.Str)); }
    inline int String::operator == (const char *s) const
                          { return (!strcmp (Str,s)); }
inline int operator == (const char *s1,
const String &s2)
              { return (!strcmp (s1,s2.Str)); }
inline int String::operator != (const String &s) const
                          { return (strcmp (Str,s.Str)); }
    inline int String::operator != (const char *s) const
                          { return (strcmp (Str,s)); }
inline int operator != (const char *s1,const String
&s2)
                          { return (strcmp (s1,s2.Str)); }
    inline int String::operator < (const String &s) const
              { return ((strcmp (Str,s.Str) < 0) ? 1 : 0); }
     inline int String::operator < (const char *s) const
              { return ((strcmp (Str,s) < 0) ? 1 : 0); }
    inline int operator < (const char *s1,const String
&s2)
{ return ((strcmp (s1,s2.Str) < 0) ? 1 : 0); }
    inline int String::operator > (const String &s) const
              { return ((strcmp (Str,s.Str) > 0) ? 1 : 0); }
     inline int String::operator > (const char *s) const
              { return ((strcmp (Str,s) > 0) ? 1 : 0); }
    inline int operator > (const char *s1,const String
&s2)
{ return ((strcmp (s1,s2.Str) > 0) ? 1 : 0); }
      inline int String::operator >= (const String &s)
const
{ return ((strcmp (Str,s.Str) >= 0) ? 1 : 0); }
    inline int String::operator >= (const char *s) const
              { return ((strcmp (Str,s) >= 0) ? 1 : 0); }
    inline int operator >= (const char *s1,const String
&s2)
{ return ((strcmp (s1,s2.Str) >= 0) ? 1 : 0); }
      inline int String::operator <= (const String &s)
const
{ return ((strcmp (Str,s.Str) <= 0) ? 1 : 0); }
    inline int String::operator <= (const char *s) const
              { return ((strcmp (Str,s) <= 0) ? 1 : 0); }
    inline int operator <= (const char *s1,const String
&s2)
{ return ((strcmp (s1,s2.Str) <= 0) ? 1 : 0); }
      inline String String::operator + (const char *s)
const
                          { return (*this + String (s)); }
    inline String operator + (const char *s1,
const String &s2)
                          { return (String
```

↪

```
(s1) + s2); }
  inline String &String::operator += (const String &s)
                          { return (*this = *this + s); }
    inline String String::operator - (const char *s)
const
                          { return (*this - String (s)); }
   inline String operator - (const char *s1,
const String &s2)
                          { return (String (s1) - s2); }
inline String &String::operator -= (const String &s1)
                          { return (*this = *this - s1); }
    inline String operator * (unsigned n,const String
&s) { return (s * n); }
   inline String &String::operator *= (unsigned nc) {
return (*this * nc); }
   inline ostream &operator << (ostream &out,const
String &s)
                          { return (out << s.Str); }
     inline unsigned len (const String &s) { return
(s.Longitud); }

          #endif

// Fin del fichero String.hh
```

Figura 2.

REQUERIMIENTOS DE STRING

Antes de ponerse a desarrollar la clase String hemos de recapacitar sobre cuales van a ser sus requerimientos más importantes. Desde nuestro punto de vista caven destacar dos:

El primero de ellos es el rendimiento; es decir, el uso de las librerías C para el manejo de cadenas podría ser engorroso pero nadie tiene duda de que era muy rápido. La nueva clase no debe, en absoluto mermar esta característica. Es más esto es un requerimiento imprescindible para una nuestro framework. El segundo, y no menos importante, es la versatilidad; es decir, el hecho de que exista un tipo para el manejo de cadenas no implica que halla de limitarme a el y que deje completamente olvidado el "chau *". La clase ha de ser capaz de trabajar con ambos.

Tampoco, por otro lado, se trata de reinventar mucho la rueda y realizar, por ejemplo, cosas como la suma de cadenas bajando al nivel de byte. En definitiva que los métodos de la nueva clase String pueden usar internamente las librerías del C.

LOS OPERADORES

Si de entre todos los métodos de la clase que se detalla en la figura 1 hubiera que destacar alguien, llamaríamos la atención sobre el grupo catalogado como operadores.

La Clase Acción

```
//=====================================
=================================//
// CLASE : Acción                    //
// DESCRIPCION : Cada uno de los actos que
pueden realizar los //
// personajes que participan en el juego sean o no //
//                 jugadores.      //
// AUTOR :          Ignacio Cea Forniés      //
//=====================================
=================================//

        #ifndef __ACCION_HH__
        #define __ACCION_HH__

        #include "Lista.hh"
        #include "String.hh"

        class Personaje;

class Accion
{

public :

//——-***          Constructores/Destructores
***————————————————————————//

Accion (const String &,Accion * = NULL,
Accion * = NULL);
void registrarVerbo (const String &);
~Accion ();

//——-***        Métodos        de        acceso
***————————————————————————————-//

static const Lista<Accion *> &acciones (void);
const Lista<String> &verbos (void) const;
int entenderVerbo (const String &) const;

//——-***        Ejecuta        la        acción
***————————————————————————————-//

static ResultadoAccion interpretar (Personaje
*,const String &);
virtual ResultadoAccion ejecutar (Personaje *,const
Lista<String> &) = 0;

protected :

Lista<String> LosVerbos;
static Lista<Accion *> LasAciones;

};

#endif

// Fin del fichero
```

Figura 3.

Cada uno de los distintos operadores sobrecargados aparece tres veces. La razón está en el segundo de los requerimientos antes impuestos a la clase (flexibilidad). No sólo ha de ser capaz de seguir entendiendo "chau *" sino que, además, debe hacerlo en un orden indiferente (vea el cuarto capítulo del curso de C++ para más detalles sobre el tema).

La Clase Coger

```
//=====================================
=================================//
// CLASE          :          Coger

//
// DESCRIPCION     :          Realiza las ope-
raciones de recogida de objetos //
// AUTOR          :          Ignacio      Cea
Fornies
//
//=====================================
=================================//

#ifndef __COGER_HH__
#define __COGER_HH__

#include "Accion.hh"

class Coger : public Accion
{

public :

//——-*** Constructores/Destructores ***————
————————————————//

Coger ();

//——-*** Ejecuta la accion ***————————
————————————————//

ResultadoAccion ejecutar (Personaje *,
const Lista<String> &);

};

#endif

// Fin del fichero
```

Figura 4.

NO ORIENTADO A OBJETOS

Hay algo (esto entra dentro de los errores que íbamos a cometer de forma voluntaria) que dentro de esta clase no tiene ningún sentido si lo miramos desde el punto de vista puro de orientación a objetos.

Podrá observar que dentro de la clase hay definidas varias funciones friend o lo que es lo mismo, funciones globales con el agravante de que, además, pueden conocer el comportamiento interno de los objetos de la clase String que manejan. Esto es una aberración contra el mundo orientado a objetos; es decir, no puedo calcular la longitud de una cadena, ¡he de pedírsela a la cadena!.

Sin embargo, se han puesto ahí con la única razón de facilitar su uso a las personas que se adentran dentro de este complejo mundo ya que estas funciones son de las más usadas para el manejo de cadenas dentro de los clásicos lenguajes procedurales. Esperamos no haber ofendido a nadie (el primer ofendido es el autor). Sólo hemos abusado un poco de que el C++ sea híbrido y no puro.

EL RENDIMIENTO

El rendimiento de los distintos métodos de la clase no se ve sólo en el funcionamiento interno de los mismos sino también en la forma en la que se comunican con el exterior; esto es, la manera en la que reciben y devuelven objetos. Un criterio general de optimización en este sentido y que, además, se ha aplicado en el desarrollo de esta clase, es, que los parámetros pasados a una clase deben ser siempre referencias en el caso de que no sean de tipos básicos del sistema y además constantes si no van a ser alteradas por el método que los recibe. Por otro lado el resultado del método debe ser siempre un objeto nuevo si no está almacenado como parte del mismo, y una referencia constante si no es un tipo básico y está almacenado como parte del mismo.

Es aconsejable, por otro lado, que si su herramienta de desarrollo se lo permite emplee un analizador de rendimiento para detectar aquellos puntos de la clase en los que el sistema pierde la mayor parte del tiempo y tratar de optimizarlos.

DISEÑANDO LAS ACCIONES

Es evidente, que un framework para el desarrollo de juegos conversacionales no puede entender inicialmente todas las distintas órdenes que cualquier usuario del mismo llegue a considerar. Hemos de buscar, por tanto una estructura de clases que permita facilidad en su mantenimiento y adaptación.

No podemos, por tanto, pensar, por ejemplo, en que la clase "Acción" que aparece en las primeras entregas de esta serie, contenga un conjunto de distintos métodos asociados cada uno de ellos asociados a un verbo distinto. Esto sería erróneo, no sólo porque podrían no estar todos los que un usuario cualquiera del framework deseara sino, además, porque decidir cual usar supone implementar más tarde o más temprano una estructura *switch* (o *if*) que no podríamos ampliar ni modificar por estar la clase finalmente implementada en una librería (*.lib*).

Descomponiendo una cadena

```
//=====================================
=================================//
// METODO          :        interpretar   //
// DESCRIPCION     :        Interpreta  y  ejecuta
una orden de usuario                  //
// REVISION        :        16/12/1995        //
// AUTOR           :        Ignacio Cea Fornies //
// PARAMETROS      :              (Personaje      *)
Personaje emisor de la orden        //
//          (const String &) Orden introducida //
// RETORNO                          :
(ResultadoAccion) Resultado de la ejecucion //
//=====================================
=================================//
ResultadoAccion   Accion::intepretar   (Personaje
*personaje,
                        const String &orden)
{
            // Crea la lista donde va a particionar la
orden introducida...
            String elVerbo;
            Lista<String> losParametros;
            String copiaOrden = trim (upper (orden));
            int terminar = 0;
            int numeroPalabra = 0;
            if (copiaOrden)
return (ResultadoAccion ("Error"));  // No hay orden

            // Trocea la orden...
            while (!terminar)
            {

// Encuentra la posicion del siguiente blanco
int posicionBlanco = find (copiaOrden," ");
                        if (posicionBlanco == -1)
                        {

                                    terminar = 1;
                                    continue;

                        }

// Si es la primera palabra es un verbo, y si no un
paramtero
                        if ((numeroPalabra++) == 0)
elVerbo = left (copiaOrden,posicionBlanco);
                        else
            losParametros.insertar
            (left (copiaOrden,posicionBlanco));

// Elimina de la cadena la palabra encontrada
                        copiaOrden -= palabra;
            copiaOrden = trim (copiaOrden);

            }

            // Considera la ultima palabra
            if ((numeroPalabra++) == 0)
elVerbo = left (copiaOrden,posicionBlanco);
            else
losParametros.insertar                    (left
(copiaOrden,posicionBlanco));

            // Busca un interprete para el verbo
Accion *laAccion = Accion::buscar (elVerbo);
            if (!laAccion)
            return (ResultadoAccion ("Error"));

            // ...y la ejecuta
            return    (laAccion    ->    ejecutar
(personaje,losParametros));
}
```

Figura 5.

UN BUEN DISEÑO

¿Cómo lo hacemos entonces?. Suponga, por un momento, que necesita traducir un documento escrito en algún idioma que no entiende. Una forma de traducirlo podrá ser la de pedir a una agencia de traductores que le buscara alguno que fuera capaz de entender ese documento y una vez que lo tuviera dárselo para que se lo devolviera en su idioma nativo.

Para la interpretación de acciones podríamos recurrir a un procedimiento semejante. Cada acción que pueda realizarse dentro del sistema va a ser una clase distinta de la que sólo existirá un objeto. Esa clase tendrá un identificador y un método de disparo de acción. Todos y cada uno de esos objetos estarán registrados dentro de una lista ordenada. Cuando el jugador introduzca un verbo por la pantalla sólo habrá que buscar a alguien de entre todos las acciones registradas que sea capaz de entender y ejecutar ese verbo y, una vez obtenido, invitarlo a funcionar.

LA AGENCIA DE EJECUTORES

¿Cómo se crea esta agencia de ejecutores?. Todas nuestras distintas acciones (verbos) van a derivar de una clase abstracta (al menos un método virtual puro) denominada "Acción".

La clase, como tal, será la que ejerza de "Agencia de ejecutores" y, para ello mantendrá una lista de objetos de tipo "Acción" un método para buscar a alguien dentro de ella. Es evidente que ambas cosas por ser de clase son estáticas.

Como base para el resto de clases "Acción" dispondrá de un método virtual puro llamado ejecutar y de un String a modo de identificador y que servirá para determinar aquel verbo que el objeto sea capaz de entender y, por tanto, para buscarlo.

Como clase, la agencia, tiene asignada una responsabilidad adicional: la de interpretar una orden del usuario.

BUSCAMOS A LOS MEJORES

- Programadores en C/C++
- Grafistas, diseñadores, dibujantes
- Expertos en lenguaje ensamblador
- Programadores 3D
- Expertos en Sonido
- Músicos con nociones de MOD, MIDI
- Programadores de juegos
- Animadores gráficos
- Infografistas con experiencia en 3D Studio, Photoshop, Deluxe Paint Animation
- Programadores con dominio de lenguajes multimedia: Authorware, Toolbook, Visual Basic, Macromedia Director, etc
- Expertos en comunicaciones

PARA DESARROLLAR

SOFTWARE MULTIMEDIA:

Presentaciones, libros interactivos, programas educativos, sistemas de comunicaciones, centros servidores de datos, videojuegos y mucho más...

SI ERES programador, músico o infografista y tienes ideas o proyectos en estudio de desarrollo ponte ya en contacto con nosotros.

TE OFRECEMOS las mejores condiciones y apoyo para producir tus programas y venderlos en el mercado nacional y extranjero. Formación en nuevas tecnologías y la mejor biblioteca de rutinas gráficas y sonido para desarrolladores.

Aportamos gráficos, rutinas o música, según las necesidades, para complementar cada proyecto.

Si estás interesado en unirte a una de las empresas más punteras en alta tecnología y desarrollo de software, no dejes pasar esta oportunidad. Envíanos carta con tus datos personales (curriculum vitae, con una muestra de tus anteriores trabajos) y un teléfono de contacto a:

DDM DIGITAL DREAMS MULTIMEDIA

DIGITAL DREAMS MULTIMEDIA
Ref. Programadores
C/ Vicente Muzas 15, 1º D - 28043 MADRID
Tf.: (91) 519.23.53 Fax (91) 413.55.77
BBS: (91) 519.75.75 Internet: ddm@servicom.es

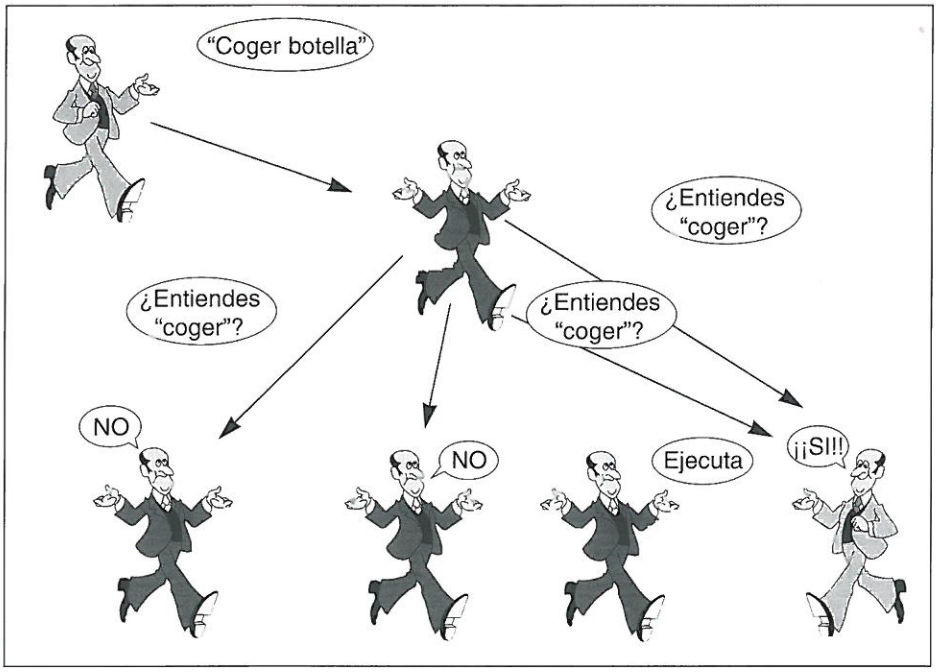


Figura 1: Los ejecutores de mandatos.

INTERPRETAR UNA ACCIÓN

Sea cual fuere la orden introducida por el usuario en la que nos fijáramos descubriríamos que tiene tres partes fundamentales: ejecutor, orden y objetos afectados.

Dado que el ordenador debe regirse para su interpretación según unos patrones muy estrictos, estamos hablando continuamente de órdenes al estilo "Tarzán"; es decir, órdenes como: "coger botella" o "lanzar piedra". En este tipo de frases es evidente que la acción es el verbo representado en la primera palabra de la cadena.

El ejecutor es quien introduce la orden, algo que estará controlado por la clase "Juego Conversacional" de la que ya vimos algo en las primeras entregas.

El verdadero problema de la interpretación de una acción consiste en discernir que objetos o personas serán afectadas por la misma. Normalmente, este alcance no se limita a sólo uno sino que abarca un conjunto de ellos. No cabe duda de que en esa lista figurará el primero el objeto o persona representado por el complemento del verbo.

Por tanto, el método que interpreta acciones (situado en el jefe de la agencia de ejecutores) funciona de la siguiente manera: recibe como parámetros el personaje ordenante y la orden por este introducida la cual será separada en verbo y complementos.

Es misión, de los ejecutores concretos decidir de que manera son afectados los objetos

Posteriormente, buscará un ejecutor adecuado para el verbo, al que pasará como parámetros tanto el personaje ordenante como la lista de objetos directamente afectados (figura 5).

Es misión, de los ejecutores concretos decidir de que manera son afectados los objetos.

¿CÓMO SE REGISTRAN LAS ACCIONES?

Como vimos en los puntos anteriores, la clase "Acción" va a ejercer de responsable de nuestra agencia de ejecutores, para lo cual dispone de una lista de sus miembros y un método que permite la localización de uno cualquiera.

Cada vez que se cree un objeto ejecutor (del tipo "Acción") debe registrarse dentro de la lista que a tal efecto dispone la clase. Dado que todas nuestras acciones van a derivar de esa clase, podemos realizar esta tarea en el propio constructor de la clase "Acción" padre.

La mejor forma de operar, por tanto, parece que es crear todas las acciones que vayan a emplearse durante el juego antes de que comience éste. No parece, por otro lado, que debido a esto se vaya a ocupar mucha memoria.

UNA ACCIÓN EJEMPLO: COGER

Como ejemplo de la forma en la que hemos de operar observe la clase "Coger" en la figura 4. Lo único que habría que hacer es redefinir el método virtual "ejecutar" y registrar todos aquellos verbos bajo los que se dispara esta acción.

LA LISTA Y EL RESULTADO NO ESTÁN

Como podrá observar, dentro del código de la clase "Acción" se manejan mucho objetos de la clase "Lista" que aún no ha sido definida. En próximas entregas le revelaremos su modo de operación pero creemos que no es imprescindible para la comprensión del código actual que usted disponga de ella. Su formación le permitirá sustituirla por alguna clase simple (mire los ejemplos iniciales del curso del C++) hasta ese momento.

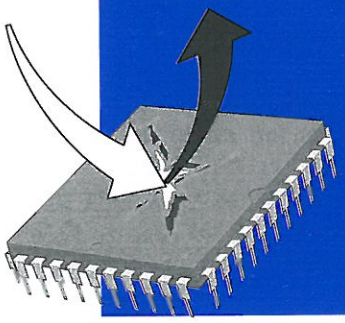
Otro tanto podría decirse de la clase Resultado cuya finalizada es agrupar el conjunto de resultados que pueden derivarse de ejecutar una acción concreta.

LA PRÓXIMA ENTREGA

En la próxima entrega nos centraremos en el gobierno general del juego y en la forma de actuar de los dioses que manejan el mundo. Aprovechamos la ocasión para desearos un feliz y próspero año nuevo.

ANALIZADORES SINTÁCTICOS

Daniel Navarro



En el artículo anterior, denominado 'analizadores léxicos', se desglosó la primera parte en el proceso de compilación de un programa. Se vio en primer lugar como construir la relación de piezas sintácticas que conforman un lenguaje de programación, es decir la formulación del conjunto de reglas léxicas que define un lenguaje, y en segundo lugar se materializó este conjunto de reglas implementando un analizador léxico capaz de, a partir de un programa escrito en el lenguaje anteriormente definido, generar un fichero que contenga la lista de piezas sintácticas que aparecen en dicho programa.

El cometido de un analizador sintáctico es simplemente comprobar si un programa concreto cumple las reglas sintácticas

En este artículo, se estudiarán las reglas sintácticas de un lenguaje de programación para pasar tras ello a implementarlas en un nuevo elemento de todo compilador: el analizador sintáctico.

El analizador sintáctico es la estructura ósea de todo compilador en la fase de análisis. El cometido de un analizador sintáctico es simplemente comprobar si un programa concreto cumple las reglas sintácticas, no genera nada, es decir, no produce ninguna salida.

Supongamos que tenemos que analizar el siguiente programa en Letra:

```
datos a = ;Un comentario
fin
```

En la primera fase de análisis (léxica) el programa queda reducido a una lista de piezas, y como todas ellas están correctamente construidas el programa anterior es correcto léxicamente. Partiendo de este conjunto de piezas, se abordará la siguiente fase de análisis (sintáctica).

```
p_datos
p_identificador (a)
p_igual
p_fin
```

El análisis sintáctico deberá concluir que este programa no es correcto sintácticamente, pues tras la pieza *<igual>* debería haberse indicado alguna constante.

LAS REGLAS SINTÁCTICAS EN NOTACIÓN BNF

Si las reglas léxicas definían la forma de los símbolos de un lenguaje, las reglas sintácticas son las que definen la forma de los programas.

Seguramente todos recuerden al profesor de lengua poniendo en evidencia a algún alumno incapaz de aprenderse sencillas normas como que toda oración está compuesta de sujeto y predicado.

Para definir las reglas sintácticas se pueden usar diferentes notaciones, como por ejemplo la notación BNF (formalismo de Backus-Naur) o los grafos.

Con las reglas sintácticas se van definiendo las secuencias de piezas sintácticas (denominadas ahora símbolos terminales) que son válidas dentro de un programa. A la secuencia de piezas o conjunto de secuencias que se define con una regla sintáctica se le da un nombre (como *<EXPRESIÓN>*) y se dice que es un símbolo no terminal.

Cuando se define el lenguaje y se dice que en un punto del programa debe venir *<EXPRESIÓN>* se está diciendo que debe venir una de las secuencias de piezas que se definen (o la secuencia que se define) con la regla sintáctica que expresa la forma que debe tomar el símbolo no terminal *<EXPRESIÓN>*.

Veamos una regla sintáctica sencilla en la notación BNF para un supuesto mini-lenguaje.

<PROGRAMA> ::= p_inicio p_fin

En esta regla *<PROGRAMA>* representa un símbolo no terminal, y *p_inicio* y *p_fin* son dos símbolos terminales.

Los símbolos terminales (o piezas sintácticas), son los símbolos que se definen con las reglas léxicas que ya se estudiaron, los reconoce el analizador léxico. Suponiendo que los símbolos terminales *p_inicio* y *p_fin* se refieran a las palabras reservadas *INICIO* y *FIN*, entonces los símbolos no terminales *<PROGRAMA>* definidos con la regla anterior deben ser todos iguales, de la siguiente forma:

INICIO
FIN

Si en dicho programa puede haber espacios, saltos de línea, comentarios, antes o después de los diferentes símbolos, o si los símbolos pueden ir en mayúsculas o minúsculas, es algo que ya se definió en las reglas léxicas y que ahora no importa.

El símbolo ::= pertenece a la notación BNF, y se usa para definir la forma de los símbolos no terminales.

Se verán ahora otros símbolos que proporciona dicha notación.

El símbolo | (pipe) representa un "o" lógico exclusivo, es decir, indica varias posibilidades de las cuales se debe dar una y solo una. La siguiente regla:
<SIGNO> ::= p_mas | p_menos

indica que el símbolo no terminal *<SIGNO>* en un programa puede ser cualquiera de estos dos símbolos terminales (o piezas sintácticas del lenguaje): el signo más o el signo menos.

El analizador sintáctico es la estructura ósea de todo compilador en la fase de análisis

Cuando se indican dentro de una regla varios símbolos terminales consecutivos quiere decir que en el programa dichos símbolos deben venir en esa misma secuencia. Una secuencia de símbolos tiene prioridad a un símbolo |, es decir, la siguiente regla:

<S> ::= A | B C

Donde *<S>* es la secuencia que estamos definiendo y *A,B,C* son tres símbolos terminales (o piezas) del lenguaje. Se entiende como:

<S> ::= (A) | (B C)
(debe venir una pieza de tipo A o dos piezas: una de tipo B seguida de una de tipo C)

Y no como:

<S> ::= (A | B) C
(primero debe venir una pieza de tipo A o una de tipo B, seguidas en ambos casos de una pieza de tipo C).

Luego, como forma terminal de *<S>* serían correctas las secuencias A y BC,

SINTAXIS DE LETRA

```
<PROGRAMA> ::= { <SENTENCIA> } p_última

<SENTENCIA> ::=
  p_etiqueta
| p_datos { p_identificador [ <DECLARACIÓN> ] }
p_fin
| <DATO> p_igual <EXPRESIÓN>
| p_si <EXPRESIÓN> { <SENTENCIA> } [ p_sino {
<SENTENCIA> } ] p_fin
| p_mientras <EXPRESIÓN> { <SENTENCIA> }
p_fin
| p_repetir { <SENTENCIA> } p_hasta <EXPRE-
SIÓN>
| p_ir p_etiqueta
| p_hacer p_etiqueta
| p_volver

<DECLARACIÓN> ::=
  p_igual <CONSTANTE>
| p_abrir_corchete p_entero p_cerrar_corchete
    [ p_igual <CONSTANTE> { p_dos_puntos
<CONSTANTE> } ]

<CONSTANTE> ::=
  p_cierto
| p_falso
| [ p_mas | p_menos ] p_entero
| p_literal

<DATO> ::=
  p_identificador [ p_abrir_corchete <EXPRESIÓN>
p_cerrar_corchete ]

<EXPRESIÓN> ::= <COMPUTO> { <OP_LÓGICO>
<COMPUTO> }

<COMPUTO> ::= <VALOR> { <OP_COMPARA-
CIÓN> <VALOR> }

<VALOR> ::= <TÉRMINO> { <OP_ADITIVO>
<TÉRMINO> }

<TÉRMINO> ::= <FACTOR> { <OP_MULTIPLICA-
TIVO> <FACTOR> }

<FACTOR> ::=
  <OP_UNARIO> <FACTOR>
|       p_abrir_paréntesis       <EXPRESIÓN>
p_cerrar_paréntesis
| <DATO>
| p_cierto
| p_falso
| p_entero
| p_literal

<OP_LÓGICO> ::= p_y | p_o

<OP_COMPARACIÓN> ::=
  p_mayor
| p_mayor_igual
| p_menor
| p_menor_igual
| p_igual
| p_distinto

<OP_ADITIVO> ::= p_mas | p_menos

<OP_MULTIPLICATIVO> ::= p_multiplicar | p_divi-
dir | p_modulo

<OP_UNARIO> ::= p_mas | p_menos | p_no
```

Cuadro 1.

y no las que define la última regla (AC y BC).

Los símbolos terminales del lenguaje que aquí estamos expresando como A, B, C ... se suelen expresar con el nombre que se les dio a las piezas sintácticas, es decir, se pondrá p_cerrar_paréntesis en lugar de ")" y se pondrá p_mas en lugar de "+", para no confundirlos con los símbolos de la notación BNF.

Los corchetes [ ] se utilizan dentro de la misma notación para definir una parte opcional, por ejemplo:

<X> ::= A [B | C] A

Define <X> como cualquiera de las siguientes secuencias:AA,ABA o ACA .

Las llaves { } se utilizan para definir una parte opcional que puede no venir, venir una vez o venir varias, por ejemplo:

<T> ::= A { B } C

Define el símbolo no terminal <T> como cualquiera de las siguientes secuencias: AC, ABC, ABBC, ABBBC, ...

Y por último resaltar que dentro de la definición de un símbolo no terminal se pueden utilizar otros símbolos no terminales, incluso el mismo de una forma recursiva. Veamos la siguiente regla:

<M> ::= A | B <M>

Pensando un poco se verá que esta regla sintáctica define <M> como cualquiera de las siguientes secuencias: A, BA, BBA, BBBA, ... Que como se puede observar equivaldría a esta otra regla:

<M> ::= { B } A

UN EJEMPLO BÁSICO

Vamos a definir con unas pocas reglas un sencillo lenguaje de programación, partiendo de los símbolos terminales (o piezas sintácticas) siguientes:

p_eof: pieza representativa del final del fichero fuente.

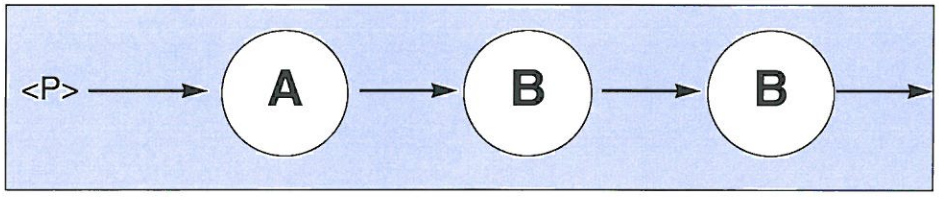


Figura 1.

p_identificador: secuencia de caracteres alfabéticos diferente a las palabras reservadas del lenguaje.

p_constante: secuencia de caracteres numéricos.

p_igual: la secuencia de caracteres := (dos puntos y un igual).

p_ptocoma: el carácter ; (punto y coma)

Si se definen las siguientes tres reglas:

<PROGRAMA> ::= { SENTENCIA } p_eof

<SENTENCIA> ::= p_identificador [p_igual p_constante] <FIN>
<FIN> ::= { p_ptocoma }

Los programas que se podrán construir serán más o menos como el siguiente:

contador ;
hora := 13
minuto := 13 ;;;
segundo := 59 ;

Una sentencia no correcta sería, por ejemplo:

décima := ;

Pues, en la segunda regla (que definía la secuencias válidas para una <SENTENCIA>) se especifica que si viene un símbolo p_igual, debe venir a continuación de este un símbolo p_constante.

Como se puede observar la potencia de este supuesto lenguaje de programación no va muy allá ... . Pero con no muchas reglas sintácticas más se puede definir un lenguaje de programación real, como Letra.

Vamos a definir con unas pocas reglas un sencillo lenguaje de programación

LIMITACIONES A LA NOTACIÓN BNF

No para todos los lenguajes que se pueden definir con esta notación se puede construir un analizador sintáctico fácilmente. Supongamos que definimos un programa <P> como:

<P> ::= <Q> | <R>

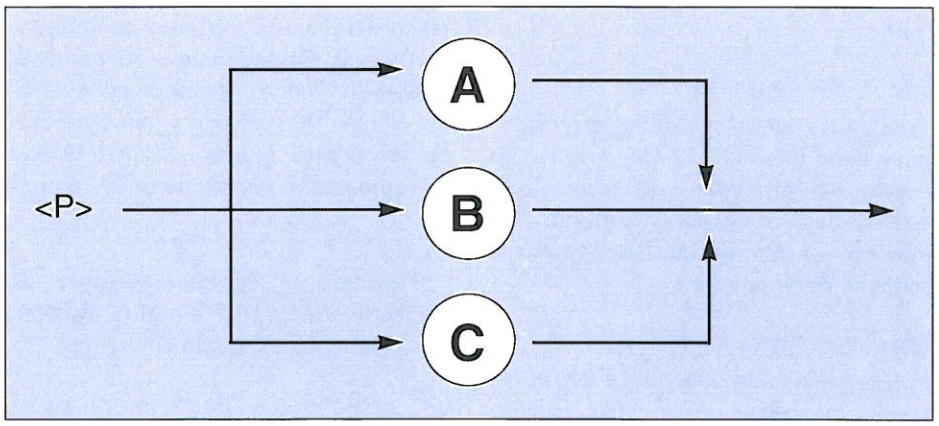


Figura 2.

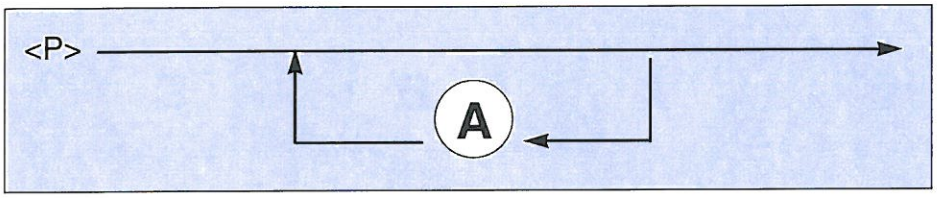


Figura 3.

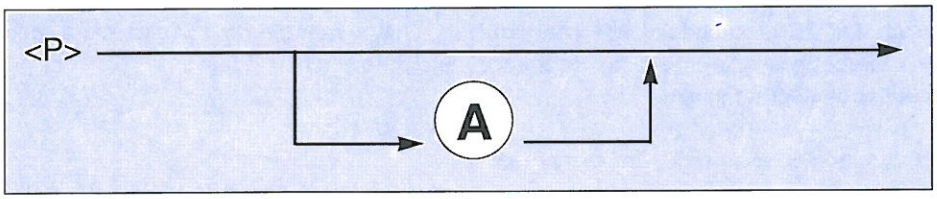


Figura 4.

<Q> ::= { A } B
<R> ::= { A } C

Estas reglas definen como secuencias de símbolos pertenecientes al lenguaje las siguientes: B, AB, AAB, AAAB, ... y C, AC, AAC, AAAC ... . Si se pretende construir un analizador sintáctico que reconozca este lenguaje, cuando pretenda analizar una secuencia como AAAC no podrá decidir que alternativa de <P> es la válida (si <Q> o <R>) analizando el primer símbolo de la secuencia (A), solo una previsión del símbolo C podrá decidir por una de estas dos alternativas. Por ello vamos a imponer la condición de que en el diseño de nuestro lenguaje todas las alternativas se puedan decidir examinado solo el primer símbolo.

El analizador sintáctico deberá concluir si el programa es o no correcto sintácticamente

Se deberá, por tanto, definir <P> como:

<P> ::= { A } (B | C)

Luego, cuando se define un lenguaje se tiene que expresar su sintaxis en reglas deterministas, de forma que examinando el siguiente símbolo que aparece en el programa sepamos que regla se debe aplicar.

GRAFOS SINTÁCTICOS

Los grafos son otra forma de representar las reglas sintácticas, mucho más visual. La transcripción de BNF a grafos no es muy compleja, basta con seguir las siguientes normas:

Las reglas del tipo *<P> ::= A B C* se pasan a grafos como el que se muestra en la figura 1.

Las reglas del tipo *<P> ::= A | B | C* se traducen en grafos como el de la figura 2.

Las reglas del tipo *<P> ::= { A }* se deben expresar como la estructura mostrada en la figura 3 en un grafo.

Y las reglas del tipo *<P> ::= [A]* se corresponden con estructuras como las de la figura 4.

Además se facilita la distinción entre símbolos terminales y no terminales, pues los símbolos terminales (recordemos, los símbolos del lenguaje o piezas sintácticas que reconoce el analizador léxico) aparecen en los grafos dentro de círculos y los símbolos no terminales (como <PROGRAMA>) se representan dentro de un rectángulo.

Veamos un sencillo ejemplo de transcripción de BNF a grafos, partiendo de la siguiente regla sintáctica:

<EXPRESIÓN> ::= <FACTOR> { p_suma <FACTOR>}

<FACTOR> ::= p_constante | p_abrir <EXPRESIÓN> p_cerrar

Donde *p_suma* es el signo más, *p_constante* una secuencia de caracteres numéricos, p_abrir el carácter de abrir paréntesis y *p_cerrar* el de cerrarlo. Estas dos reglas definen *<EXPRESIÓN>* como secuencias de símbolos del tipo:

38
(1)
(((99)))
1+(2+(33))
((6+1)+(7+0))

En la figura 5 se muestran esas dos mismas reglas sintácticas en forma de grafo.

CONSTRUCCIÓN DEL ANALIZADOR

Una vez definido el lenguaje, con BNF o con grafos, se puede construir fácilmente un analizador sintáctico para él; recordemos que este analizador simplemente comprobará si un programa respeta la estructura del lenguaje, no produciendo, por ahora, ninguna salida.

Cuando se construyó en analizador léxico este generaba una lista de piezas en un fichero a partir del programa fuente, pero ahora se verá que el analizador léxico trabaja realmente en paralelo con el sintáctico, siendo este último el que se encarga de irle pidiendo al analizador léxico las piezas, una a una según se van necesitando.

Partiendo de una sintaxis tan sencilla como pueda ser:

<PROGRAMA> ::= p_inicio p_fin

Se puede construir un procedimiento para analizar un *<PROGRAMA>*, construir por tanto el analizador sintáctico, que podría denominarse *an_PROGRAMA* de la siguiente forma:

```
void an_PROGRAMA (void)
{
  an_p_inicio();
  an_p_fin();
}
```

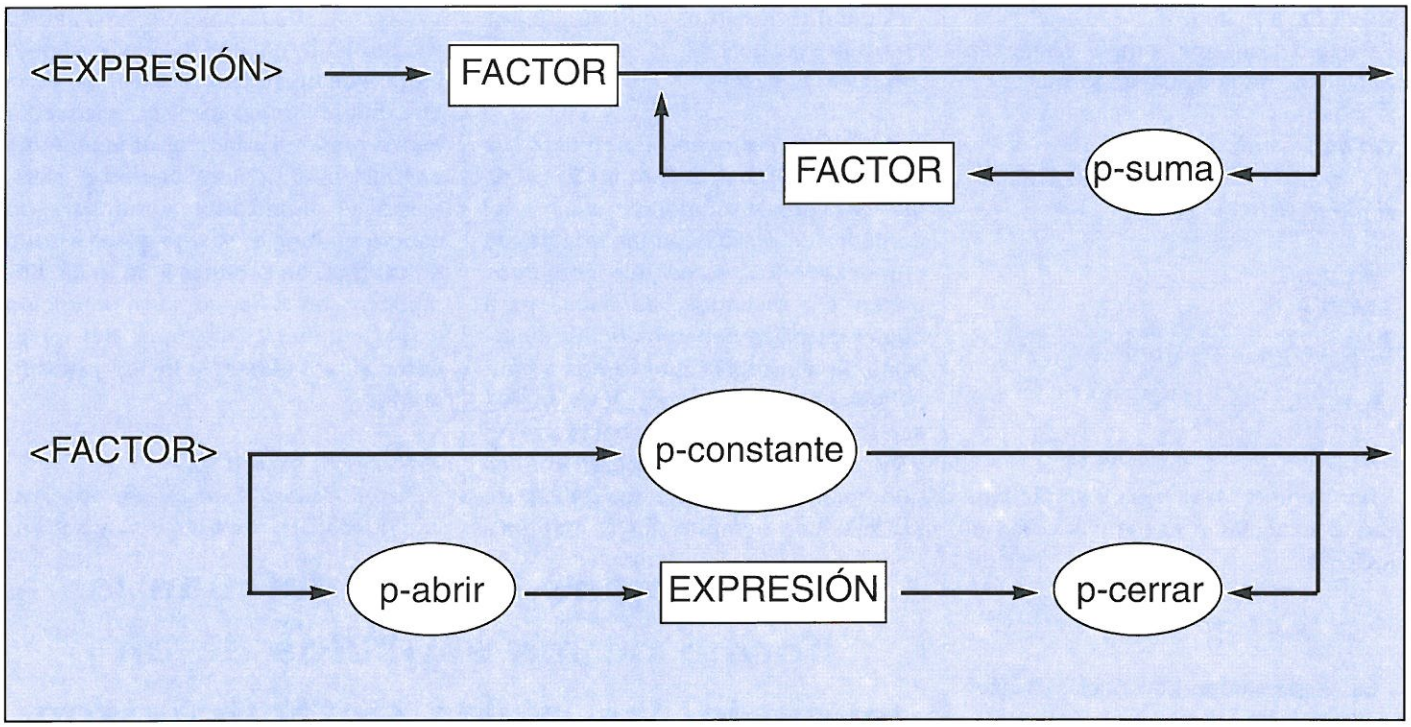


Figura 5.

Para ello se necesitan varias cosas, primero recordemos que el analizador léxico proporcionaba dos servicios que eran; primero *inicializar_léxico(fichero)* al que se debe llamar en primer lugar para indicarle el fichero fuente que vamos a analizar y el segundo *obtener_pieza()* que devolvía en una variable denominada *"pieza_actual"* un código que se correspondía con el siguiente símbolo que proviniera del fichero de entrada (a estos códigos se les denominaba *p_datos*, *p_falso*, *p_mientras*, ...)

Por tanto, antes de llamar a *an_PROGRAMA()* se debe inicializar el analizador léxico y entonces llamar a *obtener_pieza()* para tener siempre leída la siguiente pieza (pieza_actual). El código de las funciones an_p_inicio() y an_p_fin() se construye más o menos de la siguiente forma:

```
void an_p_inicio(void)
{
        if    (pieza_actual==p_inicio)
obtener_pieza();
  else error();
}

void an_p_fin(void)
{
if (pieza_actual==p_fin) obtener_pieza();
  else error();
}
```

Como se puede apreciar, el código es realmente simple; si la pieza que viene en el fichero es la que debería venir, entonces todo va bien y leemos la siguiente pieza del fichero para poder seguir con el análisis. En el caso de que venga otra pieza diferente a la que debería venir, se debe emitir un error (un error de sintaxis o "Syntax error"). Por ahora, la función que emite un error abortará el proceso de compilación, por lo que dicho proceso solo llegará a su fin cuando el programa de entrada sea correcto sintácticamente.

Bien, esta técnica la aplicaremos siempre para aplicar reglas del tipo:

<P> ::= p_a p_b p_c ...

Donde con p_a, p_b y p_c hacemos referencia a piezas sintácticas cualesquiera, simplemente se comprueba que haya venido la pieza correcta y entonces se obtiene la siguiente. Cuando se tiene que aplicar reglas con varias alternativas como:

<P> ::= p_a p_b | p_b | p_c p_a

Se construye una sentencia condicional (if ... else if ... else ...) o selectiva como la siguiente:

Dentro de la definición de un símbolo no terminal se pueden utilizar otros símbolos no terminales

```
void an_P(void)
{
 switch(pieza_actual)
 {
  case p_a: an_p_a(); an_p_b(); break;
  case p_b: an_p_b(); break;
  case p_c: an_p_c(); an_p_a(); break;

  default: error();
 }
}
```

Si las reglas que pretendemos implementar tienen una parte opcional (lo que se indica en la notación BNF poniéndola entre corchetes) como la siguiente:

<P> ::= [p_a p_b] p_c

Bastará con una simple sentencia condicional de la siguiente forma:

```
void an_P(void)
{
 if (pieza_actual==p_a)
 {
  an_p_a();
  an_p_b();
 }

 an_p_c();
}
```

Por último, las reglas sintácticas correspondientes a esquemas como el siguiente:

<P> ::= { p_a p_b p_c }

Se implementan con bucles "while" de la siguiente forma:

```
void an_P(void)
{
 while (pieza_actual==p_a)
 {
  an_p_a();
  an_p_b();
  an_p_c();
 }
}
```

Cuando dentro de la definición de un símbolo no terminal se utilice otro símbolo no terminal, como en:

<P> ::= p_a <Q>

En la implementación del analizador sintáctico esto se traducirá en una llamada al procedimiento de análisis de dicho símbolo no terminal.

```
void an_P(void)
{
 an_p_a();
 an_Q();
}
```

OBSERVACIONES IMPORTANTES

Posteriormente los procedimientos de análisis sintáctico se ampliarán para tratar el resto de la información que se necesita para que el compilador produzca una salida.

Cuando debamos aplicar reglas como la siguiente:

<P> ::= <Q> | <R>

Se debe conocer el conjunto de piezas (o símbolos terminales) con las que puede comenzar <Q> y el conjunto de piezas con las que puede comenzar <R>, estos dos conjuntos deben ser disjuntos, es decir, para cualquier pieza debemos poder determinar la alternativa que vamos a considerar para llamar a an_Q() o an_R() según sea el caso. Nos bastaría con conocer el conjunto de piezas con las que puede comenzar cualquiera de las dos, por ejemplo <R>, y entonces cuando se deba analizar <P> llamaremos a an_R() si la siguiente pieza está dentro de dicho conjunto, o a an_Q() en caso contrario.

De la misma forma, si se tiene que aplicar una regla del tipo:

<P> := [<Q>] <R>

Para ver si en el programa viene o no viene <Q> (pues es opcional) se pueden hacer dos cosas; mirar que la siguiente pieza esté o no dentro del conjunto de piezas por las que puede comenzar <Q>, o bien mirar si lo está dentro del conjunto de piezas con las que puede comenzar <R> (recordemos que no puede haber ninguna pieza con la que puedan empezar ambos).

La función error() que se implementa para los errores sintácticos, suele ser una función que nos de un error del tipo: "En la línea N del programa ha llegado una pieza tipo A y se estaba esperando que llegara una pieza tipo B". El compilador de Letra finalizará su ejecución si se encuentra un error sintáctico.

Cuando se producía un error léxico (un símbolo mal construido, como un literal que no acabara en la línea en la que había empezado) el analizador léxico respondía indicando que había encontrado una pieza de tipo p_error, como el analizador sintáctico no espera en ningún momento una pieza de tal tipo, se producirá un error sintáctico cuando llegue, pero la función error() deberá informar del error como si fuera un error léxico y no sintáctico.

PRÓXIMO NÚMERO

En el cuadro 1 se puede observar la sintaxis BNF de Letra, nuestro lenguaje. Esta sintaxis es un poco más compleja que los ejemplos que se han visto en este artículo, debido a que define más elementos, por ello se recomienda su estudio con un poco de detenimiento en los detalles. No sería mala idea intentar construir los grafos equivalentes a dicha sintaxis.

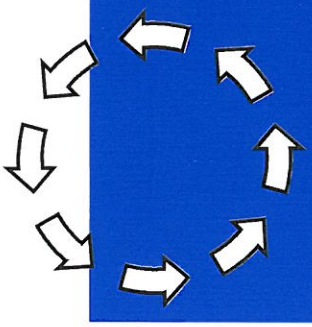
Si las reglas léxicas definían la forma de los símbolos de un lenguaje, las reglas sintácticas son las que definen la forma de los programas

En el disco de la revista se puede encontrar el analizador sintáctico de Letra (an_sinta.cpp) construido en base a esta sintaxis y que funciona en conjunto con el analizador léxico (lexico.cpp) que ya se había construido. Como no disponemos de más espacio en esta entrega, no podemos entrar en más detalles, por lo que remitimos al código incluido en el disco solo a quienes se consideren personas de espíritu aventurero.

El próximo mes se analizarán más en profundidad algunos detalles de esta implementación para entonces introducir la siguiente fase, el análisis semántico.

CREACIÓN DE DLLs PARA ARCHIVOS DE AYUDA

Enrique Castañón



El sistema de ayuda de Windows no permite incluir gráficos de más de 16 colores en los archivos de ayuda, sin embargo se deja la puerta abierta, para que estas y otras opciones, como incluir animaciones, sonidos y cualquier otra cosa que no soporte el Sistema de Ayuda de Windows, pueda ser implementado mediante la creación de Librerías de Enlace Dinámico.

DÓNDE SITUAR LAS DLLS

El hecho de registrar una función de una DLL en el archivo de proyectos de la ayuda no implica la carga de ésta, la ayuda de Windows carga la DLL solamente cuando la necesita. El Sistema de Ayuda de Windows busca la DLL cuando necesita cargarla en el siguiente orden:

- En el directorio del archivo de ayuda HLP.

- En el directorio actual MS_DOS.

- En el directorio de Windows.

- En el directorio SYSTEM de Windows.

- En el directorio que contiene el programa WINHELP.EXE.

- En la lista de directorios de la variable de entorno PATH.

- En el directorio especificado el WINHELP.INI

Si el Sistema de Ayuda de Windows no encuentra la DLL se visualizará un mensaje de error, puede definir este mensaje en la sección [FILES] del archivo WINHELP.INI, las entradas en esta sección tienen el siguiente formato:

archivo = camino, mensaje

Donde *archivo* es el archivo HLP o DLL que se quiere localizar, *camino* es el path donde se encuentra el archivo, hasta un máximo de 256 caracteres, *mensaje* es un mensaje de un máximo de 50 caracteres, que se mostrará en el caso de no encontrar el archivo. El total de cada entrada no deberá exceder de los 512 caracteres. El siguiente ejemplo buscará la DLL en el directorio C:\HLPDLL si no la encuentra, mostrará un mensaje al usuario indicando que debe reinstalar el programa:

MIDLL.DLL = C:\HLPDLL, No encuentro MIDLL.DLL, reinstale la aplicación.

Para modificar el archivo WIN-

Las librerias de enlace dinámico nos permiten aumentar la funcionalidad de los archivos de ayuda

Las librerías de enlace dinámico nos permiten aumentar la funcionalidad de los archivos de ayuda, como ya se vio en la pasada entrega, en esta se profundizará en la creación de librerías de enlace dinámico para el sistema de ayuda de Windows.

FIGURA 2

Variable	Formato	Descripción
hwndApp	U	Handle a la ventana principal del archivo de ayuda
hwndContext	U	Handle a la ventana activa actual. (Puede ser la ventana principal o una ventana secundaria)
qchPath	S	Path completo del archivo de ayuda HLP
qError	S	Puntero a una estructura que contiene información del ultimo error.
lTopicNo	U	Número de tema en el orden dispuesto al construir el archivo de ayuda.
hfs	U	Handle al archivo de ayuda.
coForeground	U	Color del primer plano.
coBackground	U	Color del fondo.

Lista de las variables internas de la ayuda de Windows.

FIGURA 3

Bandera	Valor	Descripción
fwMERR_ABORT	0x0001	Permite la opción Abortar (Por defecto).
fwMERR_CONTINUE	0x0002	Permite la opción Continuar.
fwMERR_RETRY	0X0004	Permite la opción Reintentar.

Estas banderas definen el modo en que la ayuda responde a los mensajes de error.

FIGURA 4

Error	Número	Descripción
wMERR_NONE	0	No hay error.
wMERR_MEMORY	1	Memoria local agotada.
wMERR_PARAM	2	Parámetro no válido.
wMERR_FILE	3	Parámetro archivo no válido.
wMERR_ERROR	4	Error general en macro.
wMERR_MESSAGE	5	Error en macro con mensaje.

Mensajes de error

HELP.INI puede utilizar la función del API de Windows *WritePrivateProfileString*.

VARIABLES INTERNAS DE LA AYUDA DE WINDOWS

La ayuda de Windows dispone de una serie de variables internas, que nos serán de utilidad para por ejemplo conocer el handle de la ventana principal de la ayuda, o el número de tema actual. La lista de éstas variables se muestra en la figura 2. La variable interna qError es un puntero a una estructura que contiene información del último error. Esta estructura está definida de la siguiente manera:

```
struct {

  WORD fwFlag;
```

FIGURA 5

```
#define STRICT

#include <windows.h>

// Definicion de los mensajes especificos de las ventanas filiales
#define EWM_RENDER     0x706A
#define EWM_QUERYSIZE  0x706B
#define EWM_ASKPALETTE 0x706C

// Definicion de CREATEINFO
typedef struct {
  short  idMajVersion;
  short  idMinVersion;
  LPSTR  lpstrFileName;
  LPSTR  lpstrAuthorData;
  HANDLE hfs;
  DWORD  coForeground;
  DWORD  coBackground;
} CREATEINFO;

// Definicion de RENDERINFO, utilizada en el mensaje EWM_RENDER
typedef struct {
  RECT rc;
  HDC  hdc;
} RENDERINFO;

// Definicion de BMPINFO, con la informacion del bitmap
typedef struct {
  HBITMAP hBmp;
  int width, height;
} BMPINFO;

// Variables globales
static char ClassName[] = "hlpDLL";
HINSTANCE hIns;
BMPINFO bmpInfo;

LRESULT CALLBACK _export WndProc(HWND hWnd, UINT Message, WPARAM wParam,LPARAM lParam)
{
  HBITMAP hBmpOld;
  BITMAP bmp;
  CREATEINFO *ci;
  PAINTSTRUCT ps;
```

```
 HDC memDC;

 switch(Message) {
  case WM_CREATE:
  // Añade un borde a la ventana

SetWindowLong(hWnd,GWL_STYLE,GetWindowLong(hWnd,GWL_STYLE)|WS_BORDER);

  // Lee el bitmap como recurso
   ci = (CREATEINFO*)((CREATESTRUCT FAR *)lParam)->lpCreateParams;
      bmpInfo.hBmp = LoadBitmap(hIns, ci->lpstrAuthorData);

  // Determina el tamaño del bitmap
  if(bmpInfo.hBmp != 0) {
     GetObject(bmpInfo.hBmp, sizeof( BITMAP ), &bmp );
    bmpInfo.width = bmp.bmWidth;
    bmpInfo.height = bmp.bmHeight;
  }
  else {
    bmpInfo.width = 0;
    bmpInfo.height = 0;
  }
  return TRUE;

 case WM_PAINT:
  // Dibuja el bitmap
  BeginPaint( hWnd, &ps );
  memDC = CreateCompatibleDC( ps.hdc );
   hBmpOld = (HBITMAP)SelectObject(memDC, bmpInfo.hBmp);
  BitBlt(ps.hdc, 0, 0, bmpInfo.width, bmpInfo.height, memDC, 0, 0, SRCCOPY );
  SelectObject(memDC, hBmpOld);
  DeleteDC(memDC);
  EndPaint(hWnd, &ps);

  return 0;

 case WM_DESTROY:
  // Destruye el bitmap
  DeleteObject(bmpInfo.hBmp);
  break;

 case EWM_QUERYSIZE:
      // Informa del tamaño del bitmap
      ((LPPOINT)lParam)->x = bmpInfo.width;
   ((LPPOINT)lParam)->y = bmpInfo.height;

   return TRUE;

 default:
      return DefWindowProc(hWnd, Message, wParam, lParam);
 }
}

BOOL Register(HINSTANCE hInstance)
{
 WNDCLASS WindowClass;

 WindowClass.style       = CS_GLOBALCLASS;
 WindowClass.lpfnWndProc  = WndProc;
 WindowClass.cbClsExtra   = 0;
 WindowClass.cbWndExtra   = 4;
 WindowClass.hInstance    = hInstance;
 WindowClass.hIcon        = NULL;
 WindowClass.hCursor      = LoadCursor(NULL, IDC_CROSS);
      WindowClass.hbrBackground      = (HBRUSH)COLOR_WINDOW;
 WindowClass.lpszMenuName  = NULL;
 WindowClass.lpszClassName = ClassName;

 return RegisterClass(&WindowClass);
}

int FAR PASCAL LibMain(HINSTANCE hInstance, WORD wDataSeg, WORD wHeapSize,LPSTR lpszCmdLine)
{

 // Registra la clase de ventana
 if(!Register(hInstance)) return 0;

 if(wHeapSize > 0) UnlockData(0);
 hIns = hInstance;
 return TRUE;
}
int FAR PASCAL WEP (int bSystemExit)
{
 return TRUE;
}
```

Código fuente de una DLL que gestiona una ventana filial.

WORD wError;

char rgchError(wMACRO_ERROR);
} QME;

El campo *fwFlags* indica de que manera responde Windows a los mensajes de error y *wErrror* el tipo de error, si *wError* es *wMERR_MESSAGE*, *rgchError* contiene el mensaje de error. En las figuras 3 y 4 muestra los valores y descripción de *fwFlag* y *wError*.

VENTANAS FILIALES (EMBEDDED WINDOWS)

Para incluir un gráfico de más de 16 colores, una animación, etc. es necesario crear una ventana dentro de la ventana de la ayuda gestionada por una DLL (Embedded Windows). El usuario no puede maximizar, minimizar o cambiar de tamaño estas ventanas, tampoco se pueden utilizar como hipergráficos, a no ser que se incluya código en la DLL que gestione los saltos, y en ningún caso el usuario tendrá acceso a estos hipergráficos con el teclado, debido a que el Sistema de Ayuda de Windows impide que la ventana filial reciba el foco del teclado.

El Sistema de Ayuda de Windows se encarga de crear, mostrar y destruir la ventana filial sólo cuando es necesario, la ventana filial se crea cuando se entra el tema donde reside y se destruye cuando se sale de él. El Sistema de Ayuda de Windows crea la ventana filial con el estilo WS_CHILD, esto hace que no sea necesario destruirla, se destruye cuando lo hace la ventana padre.

Para crear una ventana filial deberá hacer referencia a ésta en el archivo RTF de la siguiente forma:

{ewX NombreDLL, ClaseVentana, AutorData}

Donde X es la alineación respecto a la ventana principal, puede utilizar "l", "r" y "c" para alinear la ventana a la izquierda, derecha y centro, respectivamente. *NombreDLL* es el nombre de la DLL que gestiona la

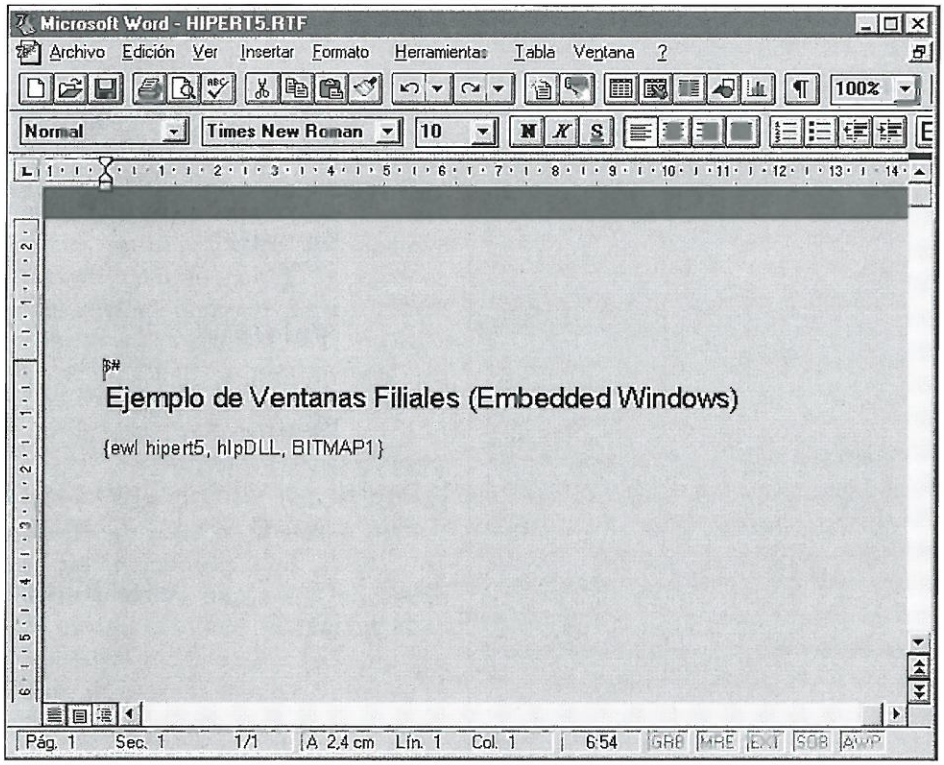

Microsoft Word - HIPERT5.RTF
Archivo Edición Ver Insertar Formato Herramientas Tabla Ventana ?
Normal
Times New Roman
10
100%
Ejemplo de Ventanas Filiales (Embedded Windows)
{ewl hipert5, hlpDLL, BITMAP1}
Pág. 1 Sec. 1 1/1 A 2,4 cm Lín. 1 Col. 1 6:54

Figura 1. Archivo RTF.

ventana filial, Windows asume que la extensión es DLL o EXE, hay que tener especial cuidado en utilizar nombres únicos, no utilice el mismo nombre para un archivo EXE y para una DLL. *ClaseVentana* es el nombre que se utilizó para definir la clase de ventana en el archivo fuente de la DLL. AutorData es una cadena que recibe el Sistema de Ayuda de Windows al crear ventana.

Cuando el Sistema de Ayuda de Windows crea la ventana, *lParam* de *WM_CREATE* contiene un puntero a una estructura *CREATESTRUCT*, el campo de esta estructura *lpCreateParams* contiene un puntero a una estructura que se deberá definir de esta forma:

```
typedef struct tagCreateInfo {
  short   idMajVersion;
  short   idMinVersion;
  LPSTR   lpstrFileName;
  LPSTR   lpstrAuthorData;
  HANDLE  hps;
  DWORD   coForeground;
  DWORD   coBackground;
} CREATEINFO;
```

Donde *idMajVersion* y *idMinVersion* identifican la versión del Sistema de Ayuda de Windows, *lpstrFileName* contiene el path completo del archivo de ayuda HLP que contiene la ventana filial, *lpstrAuthorData* contiene un puntero al tercer parametro de ewl, ewr o ewc especificado el archivo RTF, hps contiene el handle del archivo HLP, *coForeground* y *coBackground* especifican el color de primer plano y fondo de la ventana de la ayuda.

Para crear una ventana filial deberá hacer referencia a ésta en el archivo RTF

Además de los mensajes normales, las ventanas filiales reciben los siguientes mensajes específicamente definidos para usar con la ventanas filiales:

- ***EWM_RENDER (0x706A)***. El Sistema de ayuda de Windows envía este mensaje para obtener información de la imagen. Esta información es útil para copiar al portapapeles o imprimir la imagen. Los valores posibles para *wParam* son *CF_TEXT* y *CF_BITMAP*.

- ***EWM_QUERYSIZE (0x706B)***. El Sistema de Ayuda de Windows envía este mensaje para obtener el tamaño de las ventana filial. El parámetro *lParam* es un puntero a una estructura de tipo POINT, es necesario que la DLL rellene los campos x e y de esta estructura con la anchura y la altura de la ventana.

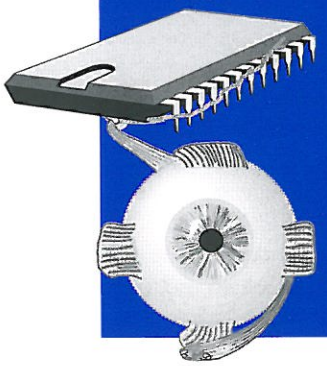
- ***EWM_ASKPALETE (0x706C)***. El Sistema de Ayuda de Windows envía este mensaje para obtener la paleta que utiliza la ventana filial, no se usan wParam y lParam.

El autor espera que esta serie de capítulos dedicados a la construcción de hipertexto hayan sido de utilidad y el agrado de todos los lectores de Sólo Programadores. A partir de ahora comienza la verdadera fase de aprendizaje, que es la que debe llevar a cabo el propio lector intentando ir más allá de lo que se ha tratado en estas entregas.

Todos los que estamos en este "mundillo" sabemos de la gran notoriedad que están alcanzando estas técnicas ante el actual auge de los sistemas multimedia, lo que está llevando al nacimiento de múltiples empresas dedicadas exclusivamente a la construcción de hipertextos e hipermedios.

EL ESTÁNDAR DPMI v.0.9

Enrique de Alarcón



El estándar DPMI (Dos Protected mode Interface) , es una librería de rutinas que funciona como extensión del DOS y que fue creada por la empresa Microsoft para poder ejecutar aplicaciones 16bits y 32bits en multitarea parcial y con capacidad para manejar hasta 4Gbytes de memoria bajo este sistema operativo (DPMI16/32bits) y la cual lo destinó a usarse como base en WINDOWS , para darle la capacidad de multitarea parcial tan apreciada por todos sus usuarios.

En general , el DPMI, se encarga de controlar la memoria (XMS) , las interrupciones , la ejecución de tareas simultáneas y la comunicación entre modo real y modo protegido dentro de las aplicaciones , aparte de otros servicios secundarios como es el de depuración. Actualmente se usa la versión 0.9 del DPMI , y éste se encuentra ya en muchos compiladores para formar parte de las librerías disponibles para el programador. En este artículo detallamos todas las funciones estándar de que dispone el programador con este extensor y de la manera de iniciar la aplicación para funcionar en modo protegido bajo el DPMI.

DIFERENCIAS ENTRE DOS/DPMI 16-32

En DOS modo real , la memoria se direcciona mediante una combinación de segmento y desplazamiento que se refiere siempre a memoria física. En total nos da 20bits de direccionamiento , que es lo mismo que decir 2^20 bytes de memoria , que son 1Mbytes. En cambio , en DPMI 16bits , los descriptores de memoria son de 24 bits , lo que da un rango de direccionamiento de 16Mbytes. A la memoria se accede mediante un selector y un desplazamiento , y el cálculo de la dirección física lo realiza el procesador valiéndose de unas tablas complementarias llamadas Tablas de descriptores (local , global...etc...).

En modo DPMI-32bits , no se usan segmentos y se referencia la memoria mediante desplazamientos lineales de 32bits , que nos da una capacidad de direccionamiento de 2^32bits , que son 4 Gb de memoria (4096 Mb) y de ahí se comprende que a partir del 80386 se disponga de un ser de registros de 32bits , puesto que son necesarios para poder hacer referencias de memoria de 32 bits..

PRIMEROS PASOS

Inicialmente siempre partimos de una aplicación que se está ejecutando en modo DOS real de 16bits (por supuesto). Lo primero que tenemos que hacer , es detectar si hay los servicios DPMI instalados y encontrar la dirección física de memoria a donde deberemos saltar para poder pasar a modo protegido. Para ello tenemos una función destinada a tal fin , y que es accesible desde modo real , es el servicio 1687h de la interrupción 2Fh. Si realizamos la llamada y todo va bien , la interrupción nos devolverá los siguientes valores: AX=0 , BX=Banderas (bit0=1 si modo 32 bits soportado) , CL=Tipo de procesador (valor 02h=80286 , 03h=80386 , 04h=80486) , DL=DPMI versión menor , DH=DPMI versión mayor , SI=Número de párrafos requeridos por el gestor DPMI (suele ser 0) , ES:DI= Dirección física a donde deberemos saltar para pasar a modo protegido. En caso de que no se ejecutase correctamente la función , AX valdría diferente a cero.

Una vez tenemos en ES:DI la dirección física de salto para pasar a modo protegido , tenemos que realizar el salto , con las banderas en AX (bit 0=1 si el programa es de 32bits) , y ES apuntando a un segmento con el espacio libre indicado en SI

Las aplicaciones en modo real puras han pasado a la historia debido a su limitación de 1 Mbytes de memoria gestionable. Actualmente , de una forma u otra , todo el mundo usa extensores del DOS para poder crear programas más potentes. Y para ello que mejor que usar el extensor más estándar del mercado , el DPMI de la casa Microsoft.

```
; Llama a la función 2Fh para obtener
; dirección de salto e información
mov ax,1687h
int 2Fh
test ax,ax
jnz NO_POSIBLE
mov MODO_PSEG,es
mov MODO_POFF,di
; Si el DPMI requiere un segmento
; con memoria para funcionar , la asigna:
cmp si,0
je NO_REQUIERE
mov bx,si
mov ah,48h
int 21h
jc NO_POSIBLE
mov es,ax
; Pasa a modo protegido
NO_REQUIERE:
sub ax,ax
call dword ptr MODO_POFF
jc NO_POSIBLE
;
; -Aqui va el código 32 bits de la aplicación.
;
; Fin de la aplicación
;
mov ax,4C00h
int 21h
```

en la anterior función , este espacio será usado por el DPMI como área de trabajo. Si se ejecuta correctamente , ahora el procesador estará en modo protegido.
Todo lo explicado , quedaría codificado de la siguiente forma:

A partir de este momento , tendremos todos los servicios del DPMI a nuestro servicio y accesibles mediánte la interrupción 31h.

Ahora debemos tener algunos datos presentes:
1-Para finalizar la aplicación que ahora tenemos en modo protegido , deberemos realizar una llamada INT 21h con AH=4Ch y en AL=código de error. Si se usa otro modo de finalizar la aplicación , no funcionará correctamente , ya que éste es el único soportado por el DPMI.
2-Todas las llamadas a servicios DPMI las realizaremos mediante la interrupción 31h. También es posible detectar antes de realizar el código anterior , una llamada a la función 1686h de la interrupción 2Fh que nos devolverá en AX si estamos en ya en modo DPMI o no (Si AX=0 , estamos ya en DPMI , si AX!=0 , no lo estamos).

FUNCIONES DISPONIBLES

Algunas normas generales: Todas las funciones devuelven el bit CF (Carry flag) a 1 si se ha producido un error y a cero si se ha ejecutado bien. Todas las funciones se llaman mediante la INT 31h , y el número de función , es el valor que debemos dar a AX antes de la llamada.

Si en una función no damos parámetros de Entrada , es que no requiere más que el número de función en AX. Si no indicamos Salida ,es que no devuelve ningún parámetro (excepto el de error en CF).

Función 0000h: Crear 1 o más descriptores en la LDT (que debe haber sido inicializada antes por el propio usuario). Entrada: CX=Numero de descriptores a crear. Salida: AX=Selector base.

Función 0001h: Libera 1 o más descriptores de la LDT.
Entrada: BX=Selector a liberar.

Función 0002h: Convierte un segmento en modo real a un descriptor usable en modo protegido. Entrada: BX=Segmento real. Salida: AX=Selector equivalente en modo protegido.

Función 0003h: Algunas funciones devuelven más de un descriptor . Se puede determinar mediante esta función el offset a añadir en la LDT para acceder al siguiente descriptor. Salida: AX=Valor a añadir para tener el siguiente selector.

Funciones 0004h-0005h: Reservadas , no usables.

Función 0006h: Devuelve la dirección lineal de 32bits de el segmento indicado. Entrada: BX=Selector. Salida: CX:DX=dirección lineal de 32bits equivalente.

Función 0007h: Cambia la dirección lineal base del descriptor indicado. Entrada: BX=Selector. CX:DX=Nueva dirección lineal para el selector.

Función 0008h: Cambia el límite del segmento indicado. Entrada: BX=selector. CX:DX=Nuevo límite del segmento.

Función 0009h: Cambia los atributos de derecho de acceso y otros del descriptor indicado por los indicados por el usuario. Entrada: BX=Selector , CL=Nuevos atributos de acceso y tipo byte , CH=Atributos extendidos de acceso y tipo byte del 80386.

Función 000Ah: Crea un descriptor con la misma base y límite que el segmento indicado. Entrada: BX=Selector de segmento de código. Salida: AX=Nuevo selector de datos.

Función 000Bh: Copia los 8 bytes del descriptor indicado de la tabla de descriptores en el buffer de 8 bytes que le indiquemos. Entrada: BX=Selector del descriptor que queremos copiar , ES:(E)DI= Puntero al buffer de 8 bytes donde copiar el descriptor indicado en BX. Salida: Buffer con la copia del descriptor.

Función 000Dh: Almacena un descriptor en la LDT. Entrada: BX=Selector.

Función 0100h: Crea un buffer de memoria mediante el DOS. Nos devuelve el segmento en modo real y uno o más descriptores equialentes al segmento DOS.Entrada: BX=Número de parágrafos (bloques de 16bytes) que queremos asignar. Salida: AX=Segmento real del buffer creado , DX=Selector equivalente para el bloque de memoria.

Función 0101h: Libera un segmento de memoria DOS modo real creado mediante la función anterior. Entrada: DX=Selector del bloque a liberar.

Función 0102h: Modifica el tamaño de un bloque de memoria DOS real creado anteriormente con la función 0100h. Entrada: BX=Nuevo tamaño en parágrafos(X*16=bytes) , DX=Selector a modificar.

Función 0200h: Devuelve el puntero a la dirección en modo real (Segmento: Offset) de donde está situado el vector de interrupción de la interrupción indicada dentro de la tabla de interrupciones real de la aplicación en que estemos. Entrada: BL=Número de interrupción de la que extraer el puntero. Salida: CX:DX= Segmento:Offset puntero al inicio del vector de la interrupción.

Función 0201h: Cambia el vector de interrupción de la interrupción en modo real que le indiquemos. Entrada: BL=Número de interrupción de la cual cambiar el vector. CX:DX=Nuevo vector Segmento:Offset (dirección modo real) de la interrupción a cambiar.

Función 0202h: Devuelve el puntero a la excepción en modo protegido de la aplicación actual que le indiquemos (Recordar que hay un rango de 00h a 1Fh excepciones). Entrada: BL=Número de excepción de la que extraer el puntero. Salida: CX:(E)DX= Selector:Offset puntero de la excepción.

Función 0203h: Cambia el puntero de la excepción indicada por el usuario. Entrada: BL=Número de excepción. CX:(E)DX= Selector:Offset nuevo para la excepción indicada.

Función 0204h: Devuelve el vector asignado en la Tabla de vectores en modo protegido a la interrupción indicada por el usuario. Entrada: BL=Número de interrupción (modo protegido). Salida: CX:(E)DX=Selector:Offset del vector correspondiente a la interrupción indicada.

Función 0205h: Cambia el vector de la interrupción en modo protegido indicada por el usuario. Entrada: BL=Número de interrupción. CX:(E)DX= Selector:Offset nuevo para la interrupción dentro de la Tabla de interrupciones.

Función 0300h: Simula la ejecución de una interrupción en modo real. Entrada: BL=Número de interrupción , BH=Banderas (Bit 0=1 indica que el controlador de INT's debe ser reseteado así como la patilla A20) , CX=Número de palabras a copiar desde el modo protegido hacia el modo real , ES:(E)DI= Selector:Offset que apunta a la estructura en modo real.

Función 0301h: Ejecuta un procedimiento escrito en modo real. Al finalizar , deberá volver con un retorno lejano (RETF). Entrada: BH=Banderas (bit0=1 si queremos resetear en controlador de INT's y la patilla A20) , CX=Número de palabras a copiar desde modo protegido a modo real , ES:(E)DI Selector:Offset de la estructura de la llamada real.

Función 0302h: Llama a una rutina escrita en modo real que finalice con IRET. Entrada: BH=Banderas (bit0=1 igual que en anteriores servicios), CX=Número de palabras a copiar de modo protegido a modo real) , ES:(E)DI= Selector:Offset de la estructura del procedimiento en modo real.

Función 0303h: Esta función se usa para obtener un único SEGMENTO:OFFSET de salto de ejecución que permita volver del modo real a modo protegido. Entrada: DS:(E)SI= Selector:Offset del procedimiento a llamar , ES:(E)DI= Selector:Offset de la estructura en modo real. Salida: CX:DX= Segmento:Offset de la dirección de salto a la que debemos ir.

Función 0304h: Libera una dirección de retorno creada con la anterior función. Entrada: CX:DX= Dirección Segmento:Offset real a liberar.

Función 0400h: Versión del DPMI. Salida: AH=Versión mayor , AL=Versión menor , BX=Banderas de estado , CL=Tipo de procesador , DH=Valor actual del PIC virtual de interrupciones , DL=Valor actual del PIC esclavo de interrupciones.

Función 0500h: Extrae información sobre los recursos de memoria del sistema. Entrada: ES:(E)DI=Selector:Offset del buffer de 48 bytes donde almacenar la información retornada. Salida: Buffer con los datos siguientes: 00h=Bloque de memoria más largo asignable , 04h=Máximo número de páginas desbloqueadas asignables , 08h=Número máximo de paginas bloqueables , 0Ch=Máximo espacio libre asignable en páginas , 10h=Número total de paginas desbloqueadas , 14h=Número de páginas libres , 18h=Numero de páginas físicas , 1Ch=Máximo espacio libre continuo asignable en páginas , 20h=Espacio de particiones en paginas , 24h-2Fh=Reservado.

Función 0501h: Crea un bloque de memoria. Entrada: BX:CX=Tamaño del bloque de memoria. Salida: BX:CX= Dirección lineal del bloque de memoria asignado , SI:DI=Bloque de memoria usado como Handle (usado por otras funciones para modificar/liberar el bloque).

Función 0502h: Libera un bloque de memoria creado con la función anterior. Entrada: SI:DI=Puntero al Handle asociado al bloque de memoria.

Función 0503h: Modifica el tamaño de un bloque de memoria ya creado. Entrada: BX:CX=Nuevo tamaño del bloque de memoria en bytes , SI:DI=Puntero al handle asociado al bloque ya creado.

Función 0600h: Bloquea la zona de memoria indicada. Entrada: BX:CX=Dirección lineal inicial de la zona a bloquear , SI:DI=Tamaño del bloque de memoria a bloquear.

Función 0601h: Desbloquea una zona de memoria bloqueada con la función anterior. Entrada: BX:CX=Dirección al inicio del bloque de memoria bloqueado , SI:DI=Tamaño del bloque memoria a desbloquear.

Función 0602h: Marcar una zona de memoria como paginable. Entrada: BX:CX=Dirección de memoria inicial del bloque a marcar como paginable , SI:DI=Tamaño de la zona de memoria en bytes.

Función 0603h: Elimina la marca de paginable a una zona marcada con la función anterior. Entrada: BX:CX= Dirección inicial de la zona de memoria a desmarcar , SI:DI=Tamaño de la zona.

Función 0604h: Devuelve el tamaño en bytes de las paginas de memoria. Salida: BX:CX=Tamaño en bytes de cada página.

Función 0700h-0701h: Reservadas.

Función 0702h: informar al sistema operativo de que un rango de páginas de memoria tienen permiso de la aplicación para pasarse a la memoria virtual. Entrada: BX:CX= Dirección inicial de la primera página a marcar , SI:DI=Número de bytes de tamaño a partir de la página indicada.

Función 0703h: Descartar el contenido de una o más páginas. Entrada: BX:CX=Dirección lineal inicial de la primera página a descartar , SI:DI=Tamaño en bytes del bloque a descartar.

Función 0800h: Acceder a una dirección física de memoria. Entrada: BX:CX=Dirección física inicial de la zona de memoria a que queremos acceder , SI:DI=Tamaño de la zona a que queremos acceder en bytes. Salida: BX:CX=Dirección lineal que puede ser usada para acceder a la memoria física.

Función 0900h: Desabilita la bandera de interrupciones virtuales y devuelve el estado anterior de la misma. Salida: AL=0 , si anteriormente estaba desactivada , 1 si estaba activada.

Función 0901h: Activa la bandera de interrupciones virtual y devuelve su estado anterior. Salida: AL=Estado anterior (1 si anteriormente estaba activado).

Función 0902h: Devuelve el estado de la bandera de interrupciones virtual. Salida: AL=Estado (1= activada).

Función 0A00h: Devuelve la dirección de llamada para usar posibles extensiones del DPMI que pueda haber instaladas. Entrada: DS:(E)SI=Puntero a una cadena Nula. Salida: ES:(E)DI= Puntero a la dirección de llamada de la extensión del DPMI , muchos registros pueden verse afectados.

Función 0B00h: Indica un punto de depurado en una dirección lineal de memoria. Entrada: BX:CX= Dirección de memoria donde situar el punto , DL=Tamaño del Punto (1,2, o 4 bytes) , DH= Tipo de Punto de depurado (0=Ejecución , 1=Escritura , 2=Escritura/Lectura). Salida: BX=Puntero al Punto de depurado.

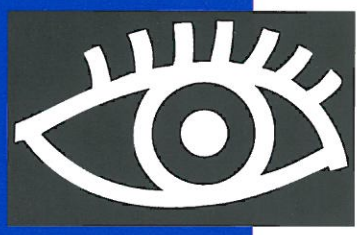
Función 0B01h: Libera un Punto de depurado creado con la anterior función. Entrada: BX=Handle del punto de depurado.

Función 0B02h: Devuelve el estado de un Punto de depuración. Entrada: BX=Handle del punto. Salida: AX=Estado de las banderas. (Bit0=1 si se ha ejecutado).

Función 0B03h: Resetea el estado de las banderas de un Punto de depurado. Entrada: BX=Handle del punto de depurado.

TODO UN ENTORNO DE DESARROLLO

Jorge R. Regidor



Visual C++ 4.0 es la nueva versión del entorno de desarrollo bajo C/C++ para sistemas Windows de 32 bits. Y decimos entorno de desarrollo y no compilador, porque este producto va más allá de lo que se espera de un compilador, e integra desde un sistema de ayuda y documentación, un depurador de código, un potente editor de fuentes y recursos, hasta una "planta de reciclaje de código".

Casi paralelamente al lanzamiento de la herramienta de programación "top-series" para Windows de Microsoft, Visual C++ 4.0, en Sólo Programadores lanzamos un curso de programación para Windows 95 basado en este compilador, en el API Win32 y en su librería de objetos Microsoft Fundation Clasess (MFC 4.0). En este curso vamos a intentar describir algunos de los aspectos más interesantes de la programación para Windows 95, además se pretende ilustrar el funcionamiento del entorno de desarrollo, indicar la localización de los temas en el sistema de documentación y ayudas de Visual C++ 4.0 e incluso trataremos tangencialmente algún punto de C++.

Visual C++ 4.0, todo un entorno de desarrollo

Visual C++ 4.0 es un entorno de desarrollo en C/C++ capaz de generar código nativo para los sistemas operativos de 32 bits de Microsoft, ya sea Windows 95 o Windows NT, y además portar dicho código sin o con muy pocas modificaciones a otro tipo de plataformas como MIPS, Alpha, PowerPC bajo Windows NT o a PowerMac y la serie 68x00 de Motorola con aplicaciones nativas para Macintosh. Además de todo esto existen herramientas de terceros fabricantes capaces de portar el código generado a sistemas UNIX. Como se puede ver poco a poco, Microsoft a través de Visual C++ 4.0 y su librería MFC se va introduciendo en muy diversas plataformas de trabajo, lo que implica que Visual C++ 4.0 puede ser tomado muy en serio para desarrollar grandes proyectos para diferentes entornos.

Son muchas las mejoras introducidas en Visual C++ 4.0, pero sin duda alguna las más llamativas residen en el cambio del entorno de trabajo. Dicho entorno se adaptada perfectamente a la nueva interfaz propuesta en Windows 95, facilitándonos el trabajo diario y haciendo éste más intuitivo y eficaz. Como ya se ha comentado con anterioridad, Visual C++ 4.0 es capaz de crear código nativo para sistemas basados en el API Win32, soporta las nuevos controles y funciones para Windows 95 y la librería MFC 4.0 se ha expandido incluyendo soporte para los componentes WOSA como Windows Sockets o MAPI (Message API), permitiendo así trabajar en redes con protocolo TCP/IP, como por ejemplo la universal Internet y enviar correo electrónico por dicha red o sobre la propia LAN al igual que hacemos con el programa Exchange de Windows 95. Otra de las novedades dentro de la librería MFC 4.0 son las clases para crear y controlar bases de datos conocidas como DAO (Data Access Objects) y como en otras versiones pero de forma extendida se soportan plantillas, gestión de excepciones, RTTI o información en tiempo de ejecución de tipos,...

Todo esto anterior, unido a la gran novedad, como es el reciclaje de código mediante la herramienta Component Gallery, de la cual hablaremos posteriormente, forma parte de los puntos a favor o novedades que se introduce en esta versión. En su contra hay poco y versa sobre los requerimientos hardware necesarios para el

correcto funcionamiento del entorno. Estos son un ordenador 486 o superior corriendo Windows 95 o NT 3.51, con un mínimo de 16 Mbytes de RAM aunque se recomiendan 20 Mbytes. La instalación típica ronda los 100 Mbytes en disco duro (con todos los ejemplos en el CD-ROM) y es necesario tener CD-ROM. Esto implica que muchos de los lectores hayan descartado la instalación del producto.

A los lectores con equipos que les permitan cumplir las especificaciones anteriores y a aquellos que aún no pudiendo a día de hoy tener posibilidades de actualizar sus equipos, pero que si quieren actualizar sus conocimientos de programación (cosa barata por otra parte) y adaptarse a lo que llega o llegará, Sólo Programadores inaugura lo que esperamos sea un curso de programación para Windows 95 interesante y entretenido.

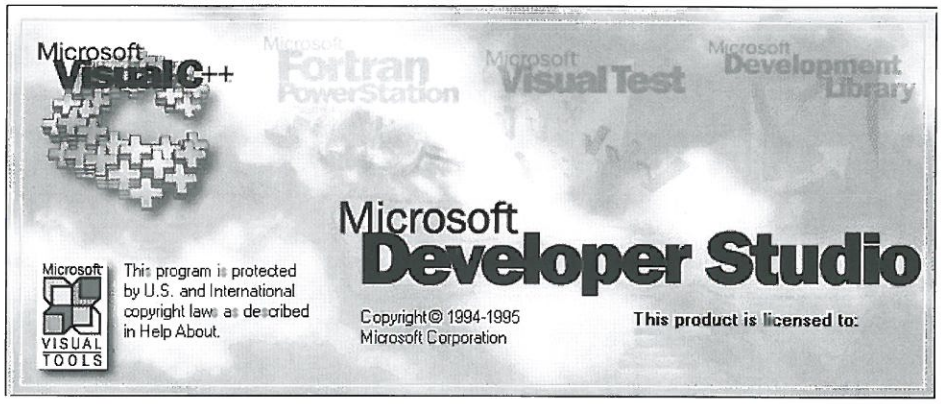

Microsoft Visual C++
Microsoft Fortran PowerStation
Microsoft Visual Test
Microsoft Development Library
Microsoft Developer Studio
Microsoft VISUAL TOOLS
This program is protected by U.S. and International copyright laws as described in Help About.
Copyright© 1994-1995 Microsoft Corporation
This product is licensed to:

Figura 1. Microsoft Developer Studio con Visual C++ 4.0

Son muchas las mejoras introducidas en Visual C++ 4.0, pero sin duda alguna las más llamativas residen en el cambio del entorno de trabajo

ORGANIZACIÓN DEL CURSO

En el curso que os presentamos queremos no sólo describir los métodos para programar en Windows 95, además queremos describir algunas de las herramientas de Visual C++ 4.0 con el fin de mejorar nuestro trabajo, queremos mostraros donde y como conseguir mayor información de un tema tanto dentro de la documentación y la ayuda de Visual C++ 4.0, sino fuera en libros de diferentes autores, plantear las diferentes formas de implementar cada nuevo paso del curso, ya sea utilizando la librería MFC o bien, programando directamente las funciones del API y por último buscar una vía de comunicación entre ambas partes con el fin de dar al curso un dinamismo que nos permita tratar los temas más interesantes para todos vosotros y no sólo los que el autor piensa que lo son.

En cuanto a los temas con los que partimos como base para realizar el curso se podrían englobar en los siguientes grupos:

* Herramientas de desarrollo visual y automático de código.
* Ayudas y documentación.
* Application Program Interface (API) de Win32.
* La librería MFC 4.0.
* Gestión de proyectos y opciones de compilación y enlazado.
* Programación en C++.

Para empezar el curso, vamos a realizar una descripción de las nuevas funcionalidades de Visual C++ 4.0 con el objetivo de mostrar a los lectores el motor principal sobre el cual se va a desarrollar el curso. En el siguiente artículo trataremos de mostrar algunas de las novedades de Windows 95 y que se deben conocer para tener una base sólida sobre la que asentar la tan famosa programación visual. Es este punto, bajo mi criterio, el más importante del curso, ya que son los cimientos de todo el edificio que intentaremos construir. El tener unos conocimientos buenos de

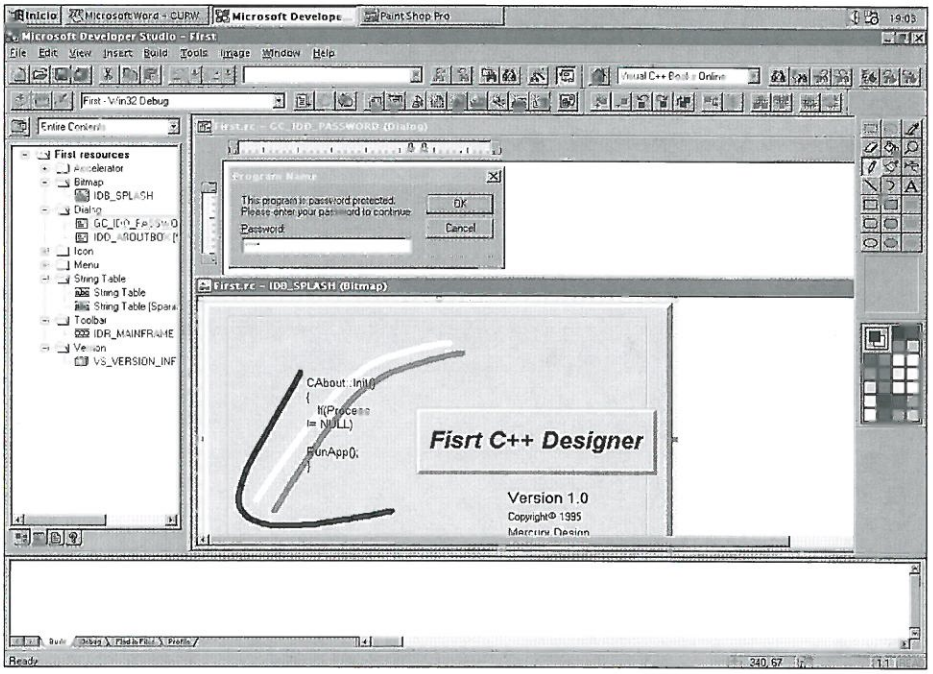


Figura 2. Visualización de los recursos del proyecto con ResourceView.

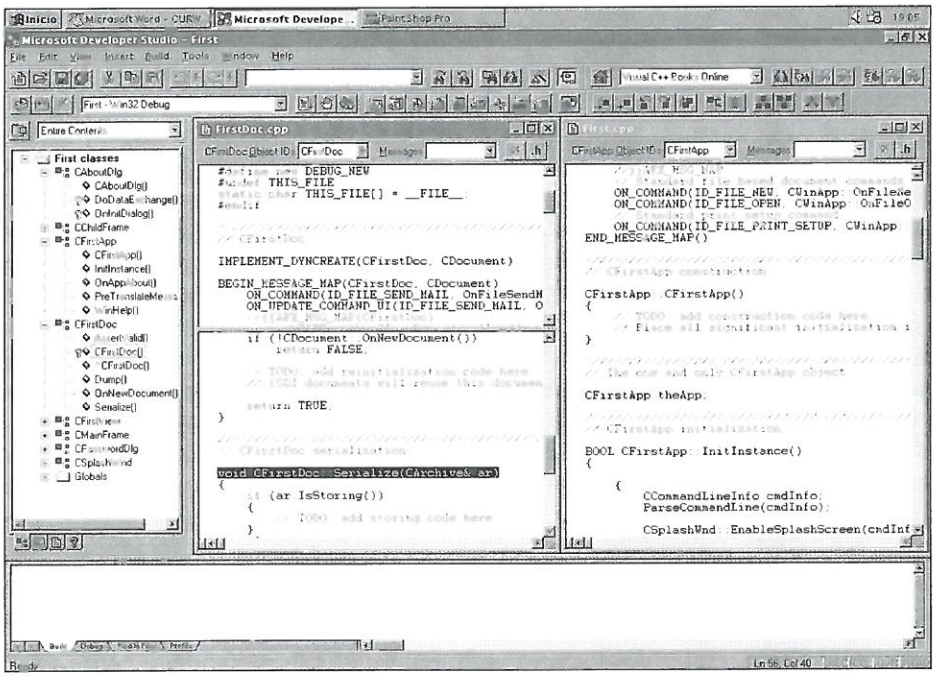


Figura 3. Visualización de las clases del proyecto con ClassView

lo que ocurre a nivel de sistema, nos librará de más de un dolor de cabeza en la programación a alto nivel. En los sucesivos artículos y con el objetivo de orientar el camino hacia la programación con la librería MFC 4.0, trataremos un tema también básico para aquellos que no tengan conocimiento del lenguaje C++. En este tema no queremos extendernos demasiado, pero si se hace necesario una introducción al lenguaje y algunas recomendaciones para la lectura. Por último, y será el núcleo del curso, se irán desarrollando los diferentes puntos de programación en Windows 95, utilizando la librería MFC y/o el API para Win32. En esta parte se tratarán diferentes temas de la programación desde la creación de la primera ventana, la inclusión de los diferentes tipos de recursos, un estudio de las posibilidades del GDI, librerías de enlace dinámico, utilización de controles de Windows 95, hasta llegar a algunos temas como la implementación de diferentes flujos de ejecución (Threads), multimedia y posiblemente algunos otros temas como MAPI, TAPI o Windows Sockets, controles OCX, OLE 2.0,... ya dentro de la programación avanzada para Windows 95.

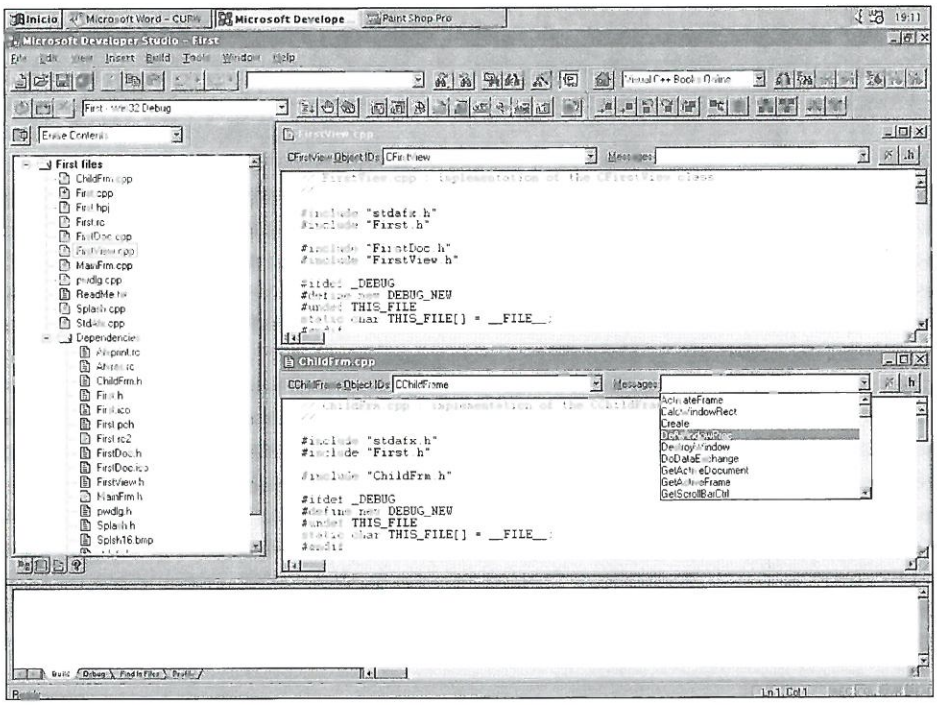


Figura 4. Visualización de los ficheros del proyecto con FileView.

MICROSOFT DEVELOPER STUDIO

Bajo este nombre reside un entorno de programación creado por Microsoft para sus diferentes entornos de programación y de documentación y comprobación, entre los que están Fortram PowerStation, Visual Test y Development Library. Como ya se ha comentado con anterioridad el entorno de desarrollo es realmente bueno y no hace falta salir de él, ni para editar los recursos de la aplicación, ni para ver la ayuda, ni para depurar el código o para consultar, por ejemplo, la revista Núm.x del mes de xxx, incluida en el Development Library de Marzo de 1996.

Todas estas mejoras además van añadidas de una serie de herramientas que nos permiten navegar por el código fuente desde varias perspectivas, por cada módulo fuente como anteriormente o bien por cada clase creada sin tener constancia de su localización modular. En los siguientes puntos vamos a hablar de algunas de estas herramientas, las cuales nos van a permitir sacarle todo el jugo a el novedoso Visual C++ 4.0.

LA VENTANA PROJECT WORKSPACE

Esta ventana es una herramienta flexible y potente que nos permite gestionar los diferentes elementos de nuestro programa de manera eficaz y sencilla. Dispone de tres paneles separados (TABS) donde se pueden visualizar los contenidos del proyecto desde tres punto de vista diferentes, como todos los recursos del proyecto (ResourceView), todas las clases del mismo (ClassView) y todos los ficheros incluidos (FileView).

Además de estas tres herramientas, se dispone de una cuarta encargada de mostrar toda la información y ayuda disponible, y permitir navegar por ella de forma eficiente.

RESOURCEVIEW

Desde esta ventana se pueden visualizar de forma jerárquica todos los recursos incluidos en el proyecto, como son iconos, menús, diálogos, bitmaps, tablas de cadenas de texto, versión,... Además haciendo doble click sobre cualquiera de ellos nos sale la ventana para poder editarlos, permitiendo así un total integración del diseño de los recursos.

FILEVIEW

Dentro de esta ventana podemos tener un punto de vista típico de los ficheros incluidos en el proyecto. Además con el botón derecho del ratón podemos compilar de modo particular cada módulo, o editar sus propiedades e incluso editar las configuraciones de compilación y enlazado del proyecto.

CLASSVIEW

Esta ventana es una herramienta muy útil permitiendo tener una imagen de todas las clases y funciones que existen en el proyecto independientemente del fichero fuente donde estén.

Desde un menú flotante se puede ir a la definición de cada clase, ver las referencias de la clase, añadir una nueva función o una nueva variable miembro, ver las clases base de las que deriva o ver las clases que derivan de ella. Además se puede añadir componentes de la galería de componentes. Es sin duda alguna una gran herramienta para navegar por las clases y miembros.

INFOVIEW

En esta ventana se visualiza toda la documentación disponible para ser consultada al tener algún tipo de duda, ya sea conocer el formato y funcionamiento de una función o bien conocer el método para utilizar por ejemplo, la especificación de telefonía asistida por ordenador (TAPI).

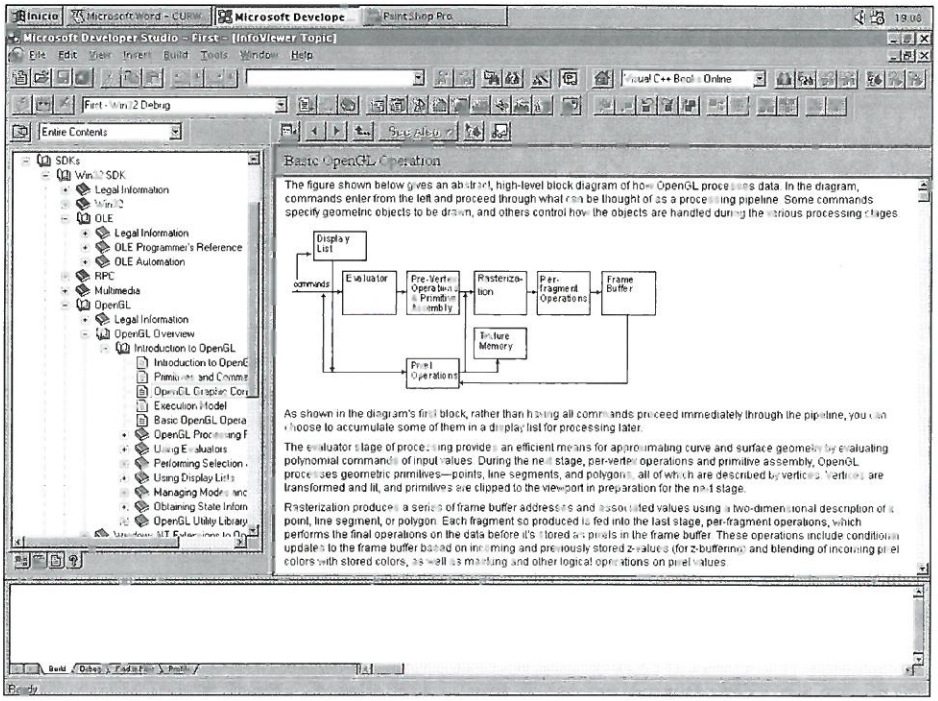


Figura 5. Visualización de la documentación del proyecto con InfoView

HERRAMIENTAS DE GENERACIÓN AUTOMÁTICA DE CÓDIGO

Sin duda alguna el mayor desarrollo de los diferentes fabricantes de compiladores tanto de C++ como de otros lenguajes, ha ido orientado en el apartado de generación automática de código y en la llamada programación visual. Estos dos puntos viajan con Visual C++ desde sus comienzos en la versión 1.0 y han ido evolucionando y mejorando hasta llegar a su zenit actual en la versión 4.0.

El método de generación automática de código empieza en la creación de proyecto mediante la herramienta conocida como AppWizard, la cual nos permite crear el esqueleto de nuestra aplicación con la sencillez de ir eligiendo opciones en las diferentes cajas de diálogo que nos van mostrando, de tal modo que al terminar, Visual C++ nos crea los ficheros fuente, los recursos, etc... necesarios para empezar a implementar nuestro código. Después de este punto, tendremos que implementar nuestros propios métodos y atender a los diferentes mensajes enviados por Windows. Para esta labor deberemos utilizar la herramienta ClassWizard que permite esto y además integrar los diálogos creados con el editor de recursos de forma visual, generando el código necesario para su manejo.

En este curso queremos describir algunas de las herramientas de Visual C++ 4.0 con el fin de mejorar nuestro trabajo

El punto nuevo viene cuando queremos incluir una nueva clase o un control OCX ya programado, ya que en Visual C++ 4.0. se incluye la herramienta Component Gallery que nos permite añadir código ya creado, compilado y probado a nuestro proyecto, así como añadir nuestros componentes para poder utilizarlos a posteriori en otros muchos proyectos.

APPWIZARD

AppWizard es una herramienta bien conocida por los programadores que hayan generado código para Windows basado en las MFCs. Básicamente se trata de un traductor que permite crear código C++ utilizando la librería MFC de clases, a partir de las diferentes opciones que le insertemos desde varias cajas de diálogo. De algún modo nosotros le indicamos que queremos hacer y él lo hace, simplificándonos el proceso de elaboración del esqueleto o base de nuestro programa.

Algo que vamos a agradecer los programadores españoles es la existencia de una versión para castellano, ya que con anterioridad, todos los mensajes y textos generados eran en inglés, teniendo el programador que traducirlos o bien tener un programa con partes en castellano y otras en inglés.

Hay un punto del cual se van a alegrar los conocedores de AppWizard y es que esta herramienta puede ser personalizada para no limitar la generación de la base del programa solamente a las opciones que AppWizard nos muestra por defecto. ¿Quién alguna vez no ha pensado que necesitaba algo que el AppWizard no podía hacer? Pues bien, ahora se puede personalizar de modo que ya si podremos hacerlo. AppWizard puede ser personalizado de tres diferentes formas:

* Crear un AppWizard propio en función del proyecto actual, de tal forma que podemos crear la base compartida por todos nuestros proyectos.
* Crear un AppWizard propio extendiendo las opciones que nos brinda el estándar creado por Microsoft. Este es un buen método para personalizar algunos pequeños puntos que siempre tenemos que retocar.
* Crear un AppWizard con todas las opciones creadas por nosotros.

CLASSWIZARD

ClassWizard es otro de los conocidos dentro de las versiones anteriores de Visual C++. Su principal labor es facilitar al programador el control y la creación de las diferentes clases de la aplicación así como permitir añadir y eliminar funciones miembro a cada clase. Además y en conjunción con el editor de recursos, nos permite crear a partir de los diálogos creados visualmente la clase encargada de controlarlos, con sus diferentes funciones y variables miembro.

Por otra parte y si nuestra aplicación soporta la especificación OLE 2.0., nos permite realizar el control de los diferentes opciones para añadir a cada clase.

El depurador de Visual C++ se puede conectar remotamente mediante TCP/IP, además de vía serie como anteriormente

WIZARDBAR

WizardBar es una barra que se añade automáticamente a las ventanas de edición de código fuente al utilizar la librería MFC. Con ella se pueden definir manejadores de mensajes, sobrecargar las funciones virtuales de cada clase derivada de una MFC y navegar por los ficheros C++ del proyecto. Con esta barra se facilita la inclusión de nuevos métodos a las clases y su acceso. De algún modo esta barra recuerda a la existente en Visual BASIC.

COMPONENT GALLERY

Esta herramienta es sin duda alguna la joya de la corona de la versión 4.0 de Visual C++ y sino que se lo digan a Quintín Aprovechalotodo estrella de la campaña de publicidad de Visual C++ 4.0. El tema no es para menos, ya que si es como parece, Component Gallery es un almacén de componentes reutilizables (¿una planta de reciclaje?) para clases C++ y controles OLE (OCX), aunque también se pueden almacenar otro tipo de código.

Cuando estemos trabajando en un proyecto y necesitemos de una clase que sabemos que está en el almacén, solamente se tiene que abrir Component Gallery y pulsar el botón de Insertar. El código fuente de la clase será añadida al proyecto para ser utilizado de inmediato.

Para el caso de controles OLE, la operación de insertar genera el código necesario para utilizarlo dentro de nuestro proyecto. Además el control OLE se añade a los controles OLE del editor de diálogos por si se quiere utilizar este control desde un diálogo.

Parece claro que con la reutilización de código podremos aprovechar mejor el esfuerzo de proyectos anteriores y como parece seguro, aparecerán no sólo controles OLE comerciales como en su día aparecieron las VBX, sino clases C++ para poder ser utilizadas. ¿Es éste un posible futuro campo de trabajo? De hecho se Microsoft incluye una serie de clases C++ y controles OLE los cuales pueden ser utilizados. La primera impresión al haber utilizado alguno de ellos es realmente interesante debido a la sencillez y rapidez con la cual se pueden añadir nuevas partes al proyecto.

Visual C++ 4.0 permite personalizar AppWizard, permitiendo al desarrollador crear sus propios cimientos de sus aplicaciones

DESARROLLO

En este apartado vamos a tratar algunas de las novedades que se incluyen en Visual C++ 4.0, en cuanto a compilación, enlazado o depuración de código. Como todos sabemos

Visual C++ es un compilador de C y C++ soportando las últimas novedades en la especificación de C++ como son RTTI, control de excepciones o namespaces.

COMPILACIÓN INCREMENTAL

Esta es una nueva capacidad que permitirá minimizar el tiempo de compilación de cada módulo fuente. Como ya sabemos, todos los compiladores sólo compilan aquellos módulos fuente que han sido modificados, Visual C++ va un poco más allá y compila sólo las funciones que hayan sido modificadas desde la compilación anterior. Esto se agradece ya que en módulos fuente con un código extenso el tiempo de compilación debe disminuir drásticamente.

ENLAZADO INCREMENTAL

Como su nombre también indica, esta nueva propiedad permite solamente enlazar aquellos módulos objeto que hayan compilado desde la última operación de enlazado. Tanto esta propiedad como la anterior parecen ser bastante útiles mejorando notablemente uno de los puntos negros de todos los compiladores.

EDITOR DE RECURSOS

En cuanto al editor de recursos hay que resaltar varios aspectos como son la posibilidad de incluir los nuevos controles de Windows 95 en el proceso de diseño de diálogos, punto necesario para desarrollar aplicaciones con verdadera interfaz para Windows 95.

Otra novedad, es la posibilidad de leer formularios de Visual BASIC e incluirlos en un diálogo. Después puede guardarse como un diálogo normal. Esto será muy útil para los desarrolladores en Visual BASIC que quieran dar el salto a C++.

DEPURADOR DE CÓDIGO

A los programadores de Visual C++ de las versiones para Windows 3.1 les sorprenderá gratamente la potencia y versatilidad del mismo. Además, existen algunos pequeños detalles realmente interesantes. Uno de estos son los llamados Data Tips que permiten ver el valor de las variables y las expresiones pasando el cursos sobre ellas. El valor se muestra como un título al lado del cursor cuando el programa está en pausa.

Además, cuando se visualiza el contenido de un array, objeto o estructura MFC, Visual C++ expande automáticamente la variable mostrando los valores de los campos más importantes campos.

Además la ventana de visualización de variables tiene nuevas propiedades para acceder a la información de las variables en cada contexto del programa. De este modo el Auto Tab permite ver las variables usadas cuando pasas por una función, Local Tab permite ver las variables locales de la función actual, this Tab permite ver el contenido del objeto apuntado por this.

Quizá lo más llamativo del depurador es la posibilidad de conectarse remotamente, ya sea por conexión serie como en otras versiones o por en LAN mediante TCP/IP. De esta forma podremos tener un programa en otro ordenador visualizando los mensajes de depuración.

UTILIDADES EXTERNAS A MICROSFT DEVELOPER STUDIO

En este apartado vamos a ver algunas las herramientas añadidas a Visual C++ 4.0. pero que no forman parte del entorno de desarrollo, Microsoft Developer Studio. Entre estas herramientas se incluyen el Help WorkShop, MFC Tracer, Microsoft Developer Roadmap, Pview, Spy++ y Windiff.

Help Workshop es un entorno de desarrollo de ayudas para Windows 95, que inexplicablemente no está integrado en Microsoft Developer Studio, ya que parece casi lo único que le falta a este entorno.

MFC Tracer es un trazador de mensajes para la librería MFC en modo de depuración, permitiendo visualizar todos los mensajes recibidos al programa o bien sólo aquellos que deseemos.

Microsoft Developer Roadmap es una utilidad que ilustra la estrategia de Microsoft en cuanto a sus nuevos productos para desarrolladores.

PView es una utilidad que permite visualizar todos los procesos cargados en el sistema, además permite acabar con cada uno de ellos. Seguro

Component Gallery es sin duda alguna la joya de la corona de la versión 4.0 de Visual C++

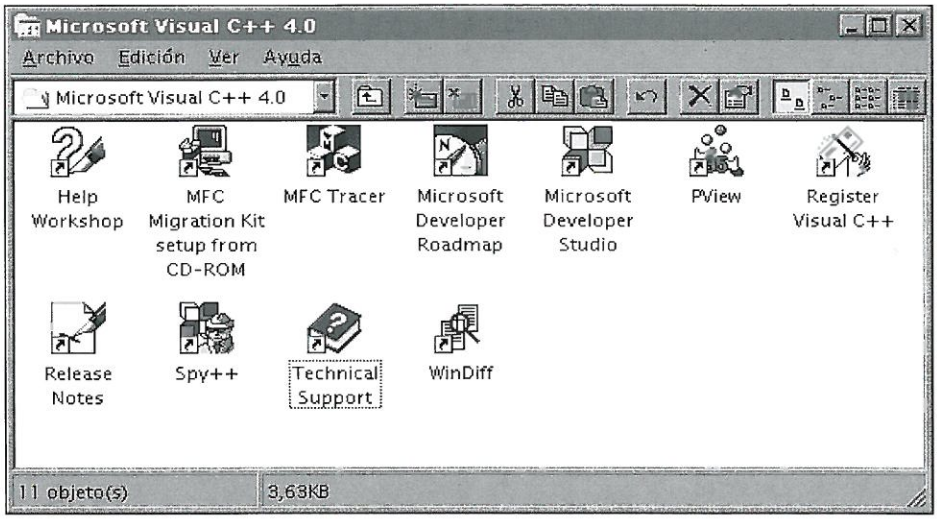


Figura 6. Grupo de programas de Visual C++ 4.0

que será utilizado más de una vez al meter la pata en un programa.

Spy++ es quizá una de las utilidades más completas, ya que permite hacer un control de todos los procesos, threads y ventanas cargados en el sistema, así como ver todos los mensajes recibidos por una ventana...

Visual C++ 4.0 introduce Proyect Workspace, una herramienta que nos permiten navegar por el código fuente desde varias perspectivas

Windiff es una herramienta que permite visualizar las diferencias entre dos ficheros de texto, lo que sin duda será utilizado cuando tengamos varias versiones de código fuente.

DOCUMENTACIÓN INCLUIDA

La documentación incluida dentro de Visual C++ 4.0 es mucha y variada, teniendo una herramienta de visualización integrada dentro de Developer Studio, permitiendo así una visión conjunta con el código fuente.

En la documentación se incluyen los ejemplos de programación sobre casi cualquier tema relacionado con Windows, se incluyen los libros completos sobre la programación en Win32, sobre la especificación OLE 2.0, sobre programación ODBC y DAO para bases de datos, una guía de diseño de entornos gráficos, se añaden los libros para programación mediante la librería MFC, tutoriales, sintaxis del lenguaje C/C++, una serie de artículos relacionados con trucos y errores de los diferentes métodos de

Además de toda esta información, el entorno de ayuda y documentación es totalmente compatible con Microsoft Developer Library, lo que permite integrar fácilmente toda la información suministrada trimestralmente en estos CD-ROM.

En el caso de los códigos fuente de los ejemplos están almacenados en el CD-ROM, pero son muy sencillos de instalar en el disco duro sólo pulsando un botón. La cantidad de ejemplos es inmensa y de todo tipo de dificultad lo que nos permitirá no tener que partir desde cero al intentar implementar algún tipo de aplicación.

CONCLUSIONES

En este artículo se ha pretendido ilustrar brevemente las pautas sobre las cuales va a discurrir el curso de programación y presentar las novedades introducidas por Visual C++ 4.0 frente a versiones anteriores del compilador. No se ha entrado en detalle de cada herramienta, ya que posteriormente y cuando sea necesario entraremos a detallar cada utilidad que debamos manejar, aunque es el propio lector el que debe indagar y probar cada utilidad.

La ventana de visualización de variables tiene nuevas propiedades para acceder a la información de las variables en cada contexto

En el siguiente articulo vamos a intentar resaltar algunas de las diferencias entre Windows 3.1 como sistema de 16 bits y Windows 95 como sistema de 32 bits (según la versión de Microsoft, bastante discutible por otra parte). Resaltaremos la multitarea preemtiva de Win32 frente a la cooperativa de Win16, los espacios de direccionamiento de 32 bits, la protección entre tareas,... Y si es posible entraremos en la creación de una aplicación mínima utilizando el API Win32.

BIBLIOGRAFÍA

En este apartado de bibliografía cabe destacar la lectura de los libros que se suministran con Visual C++ 4.0., y en concreto los siguientes:

* Win32, dentro de SDKs y WIN32 SDKs.
* Programmer's guide to Windows 95 dentro de SDKs, WIN32 SDKs,

GUIDES.
* MFC 4.0 dentro de Visual C++ Books.
* C/C++ dentro de Visual C++ Books.
* y como no Tutorial dentro de Visual C++ Books.

Además y ya en papel, para aquellos que prefieran otros autores o bien no les gusta la lectura en pantalla, se recomienda:

* Microsoft Win32 Programmer's Reference.
Varios Autores.
Microsoft Press. (Inglés)

Para los programadores que no conozcan la programación para Windows, y aunque es un libro para Windows 3.x, les recomendaría:

* Turbo C++. Desarrollo de programas en Windows 3.x.
Jaime Peña Tresancos.
Editorial Anaya Multimedia

Para aquellos que no conozcan la programación en C++, también les recomiendo un par de libros como son:

* Programación Orientada a Objetos en C++
Ricardo Devis Botella.
Editorial Paraninfo.

* El C++. Lenguaje de programación.
Bjarne Stroustrup.
Addison-Wesley/Díaz de Santos.

CORREO DEL LECTOR

En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

Hola amigos, primeramente quisiera felicitarles por la revista, que sirve de ayuda a muchos desarrolladores de software independientes como yo. Mi duda es la siguiente: estoy realizando una aplicación en modo gráfico (640 x 480 y 256 colores) y los módulos BGI de Borland me permiten esa resolución, pero sólo con 16 colores. Soy capaz de trabajar con más colores gracias al artículo publicado en el número 12 sobre el estándar VESA para SVGA, pero mi gran problema es la representación del ratón. ¿Cual sería la mejor solución al problema?

José Juan Mora Pérez
(Huelva)

La solución más sencilla es, desde luego, la de conseguir un módulo BGI que soporte modos SVGA y que además gestione el redibujado del cursor. Pero si quiere desarrollar usted mismo toda el manejo del entorno, entonces debe programarse un completo *driver* para el ratón, aunque también es posible encontrarlos ya creados. Debe empezar creando una rutina que realice los siguientes pasos:

1.- Redibujado de la parte de pantalla modificada al plantarse el cursor. La primera vez que se llame a la rutina del ratón este paso no se realizará, pues todavía no se habrá escrito en pantalla ninguna vez.

2.- Llamada al *driver* del ratón para conocer su nueva posición y el estado de sus botones (pulsados o no pulsados). Preferiblemente se ejecutarán sus funciones de más bajo nivel: la que informa del estado de los botones y del desplazamiento relativo efectuado.

3.- Copia en memoria de la porción de pantalla que se verá afectada al dibujarse el cursor en la nueva posición.

4.- Dibujado del cursor. Debe realizarse como si fuera un *sprite*, transparentando o no dibujando los *pixels* del color 0. En los modos gráficos de alta resolución, en los que la pantalla se compone de más de un segmento (los que le interesan), se debe vigilar que al dibujar cada línea del cursor, sea necesario cambiar de segmento y realizarlo en consecuencia, preferiblemente mediante el estándar VESA.

Una vez programada y probada la rutina, sólo resta "engancharla" a una interrupción que se ejecute cada vez que se redibuje la pantalla, esperando al retrazado vertical para conseguir la mayor suavidad y la total eliminación de "parpadeos".

Llevo comprando su/nuestra revista desde su primera aparición y cuando, en el número 8, empezaron a publicar las fichas de programación, mi valoración global de Solo Programadores subió muchos puntos y me parecieron una fantástica idea. Pero recientemente observé que se dejaron de editar, ¿a que se debe esta decisión?, ¿era solo un paréntesis y van a continuar con la colección?

Eloy Mier Pérez

Tiene razón, durante unos meses no hemos incluido las prácticas fichas que regalábamos con la revista, pero era debido a motivos editoriales y de marketing, ya que entregábamos otros componentes con ella. La edición de las fichas vuelve a reanudarse y, esperemos, no se detendrá hasta acabar con la colección. Además se están desarrollando otras colecciones simultáneas o intercaladas, como por ejemplo un completo curso en forma de fascículos sobre el famoso y potente compilador Visual C de Microsoft, que verá la luz en breve.

No se preocupe, el archivador para las fichas se continuará rellenando.

Quiero realizar un proyecto con imágenes digitalizadas y programarlo bajo Windows, pero no sé que compilador utilizar: Visual Basic, Visual C, Delphi u otro. ¿Cual es el mejor? También me gustaría introducir unas imágenes de vídeo dentro de un programa, pero ignoro su complejidad. Si pudieran aconsejarme al respecto, les estaría muy agradecida.

Belén V.P.
(Mérida/Badajoz)

R Cualquiera de los compiladores que menciona le permitirán realizar su objetivo, la elección dependerá de su experiencia en desarrollos bajo Windows. Si dicha experiencia es escasa, lo mejor es que utilice Visual Basic, pues es un poco más sencillo e intuitivo que los otros; por contra, su velocidad de ejecución es sensiblemente menor que la de Delphi. Programarlo mediante Borland C o Watcom C requerirá más experiencia y tiempo, pero el resultado será más potente.

En la actualidad, el incorporar imágenes animadas (provenientes de vídeo o generadas infográficamente) o complejas melodías y sonidos en los programas es bastante sencillo, Windows (y Windows 95 mejor todavía) incluye librerías, DLLs y servicios que facilitan el manejo de los procesos multimedia. En ocasiones, basta con sentencias del tipo: *start_video(), next_track()* o similares. Para encontrar información técnica al respecto, todos los compiladores incorporan ficheros de ayuda entre los cuales suele encontrarse uno con el nombre *multimedia* o similar. Pero para aprender a manejar las APIS o librerías lo mejor es un buen libro (moderno) de programación en Windows o seguir los artículos sobre multimedia que se incluyen es la revista.

P *Hola, últimamente estoy trabajando a la vez con dos o tres compiladores simultáneamente y realmente es un engorro el apagar y encender el ordenador cada vez que cambio de paquete (por las definiciones de las variables del sistema necesarias para el funcionamiento de cada uno de ellos). Además esto me obliga a mantener muchos tipos de arranque. He probado a mantener definidas todas las variables conjuntamente (SET INCLUDE = C:\BC4\INCLUDE; C:\WATCOM\INCLUDE; C:\TASM\INCLUDE), pero siempre encuentran el fichero equivocado y se arman un lío. ¿Alguna sugerencia? Gracias.*

Pedro García
(Castellón)

R Se suele utilizar un sencillo truco con el cual nunca existen "incompatibilidades". Consiste en no incluir en el arranque las definiciones y crearse los ficheros BAT (o CMD) necesarios para cada uno de los compiladores.

Por ejemplo:

```
INIBC4.BAT

SET OLDPATH = %PATH%

SET PATH = %PATH%; C:\BC4\BIN

SET INCLUDE = C:\BC4\INCLUDE

SET LIB = C:\BC4\LIB
```

Obsérvese que antes de modificar el estado inicial de la variable PATH, se guarda su valor en OLDPATH, para tenerlo disponible, y restaurar el sistema mediante el fichero:

```
DESINI.BAT
SET PATH = %OLDPATH%
```

Con todo esto, podemos cambiar de un compilador a otro mediante la secuencia:

```
INIBC4 ; Sistema preparado para Borland C
...
DESINI
INIWAT ; A trabajar con Watcom
```

También se puede unir la "restauración" del sistema a los ficheros INI, incluyendo en la primera línea:

CALL DESINI.BAT con lo que se acelera aun más el cambio de plataforma de desarrollo.

P *Intento aprender el lenguaje C de manera autodidacta, pero me choco con dudas "insalvables". ¿Porqué en los fuentes en C, se escriben las sentencias #include de dos maneras diferentes: <fichero.h> y "fichero.h"?*

Mariano Fuensanta
(Barcelona)

R Pregunta sencilla (¡que bien!). Cuando se usan los símbolos <> es porque se refiere a los *includes* (librerías) estándar o propias del compilador y el preprocesador irá a buscarlas a los directorios definidos por defecto (los directorios *INCLUDE* que se encuentran en el directorio del compilador); y cuando se utilizan las comillas "" los buscará en el directorio activo en donde se encuentre el resto de los fuentes. Sólo se altera la forma de buscar los ficheros, el resto del tratamiento es idéntico.

ENTRE EN EL MUNDO DE

SUPEROFERTA PARA SUSCRIPTORES Y LECTORES DE

SÓLO PROGRAMADORES
Revista práctica para usuarios de PC

+ 50% DESCUENTO en VISUAL C++

CON LAS HERRAMIENTAS VISUALES DE MICROSOFT

PARA BENEFICIARSE DE ESTA OFERTA LOS LECTORES NO SUSCRIPTORES DEBERÁN REMITIR LOS CUPONES QUE APARECERÁN EN ESTA PÁGINA

CURSO DE VISUAL C++

CUPÓN 1

¿Qué sería de la más potente herramienta de desarrollo sin el genio para el reciclaje de Quintín Aprovechalotodo?

Si Quintín Aprovechalotodo fuera un desarrollador, estaría orgulloso. Porque el nuevo Microsoft® Visual C++ 4.0™ se basa en los mismos principios para los que él vive: Nunca empieces de cero.

¿Quién es Quintín Aprovechalotodo? Un genio del reciclaje que en cada objeto usado ve una oportunidad para demostrar su brillantez creativa. De la misma manera que podría hacerlo un desarrollador si utilizara el nuevo Visual C++ 4.0. Ninguna otra herramienta de desarrollo ofrece tantas posibilidades de reutilización, lo que le permite generar aplicaciones más robustas en menos tiempo.

Ahora ya puede reciclarlo casi todo, pero no necesita ser un genio para ello.

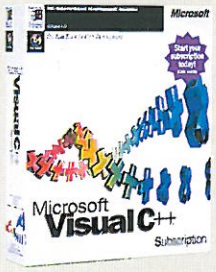
El nuevo Component Gallery le permite guardar y reutilizar sus propios componentes de C++, controles OLE, así como los componentes de terceras partes. Y, las MFC 4.0 le proporcionan más de 150 clases y 120.000 líneas de código que ya no tendrá que escribir ni comprobar.

El nuevo Developer Studio facilita el que la reutilización sea algo intuitivo. Con el ClassView se pueden visualizar las relaciones entre clases. Y con un solo click tendrá acceso a la Development Library de MSDN, Microsoft Visual Test, Fortran PowerStation, y a la versión de control de versiones Microsoft Visual SourceSafe™.

Y por supuesto, soporta el lenguaje extendido de C++, incluidos los espacios en el nombre de las variables y RTTI, lo que le aporta aún más flexibilidad.

Ahora que ya se ha formado una idea más clara de las posibilidades del reciclaje, usted puede comenzar con la suscripción a Visual C++, que incluye el nuevo sistema de desarrollo Visual C++ 4.0 más tres Revisiones para Suscriptores que le enviamos automáticamente.

Para más información acerca de la Suscripción a Visual C++ 4.0, o para saber cuál es el Distribuidor Microsoft más cercano basta con llamarnos a Microsoft al (91) 804 00 96, o bien ponerse en contacto con cualquiera de los Distribuidores que aparecen más abajo. ¿Quién sabe?, quizá tu también acabes pensando como Quintín Aprovechalotodo.



Acceso rápido mediante el Component Gallery a los controles OLE y a los componentes reutilizables de C++.

Librería de Clases MFC 4.0 añade el soporte necesario para los controles de Windows® 95.

Soporte Cliente-Servidor. Motor de datos Jet integrad para el acceso a bases de datos y soporte ODBC para acceder a datos remotos.

Custom AppWizard. Ahora podrá cear su propio AppWizard para las necesidades específicas de s aplicación.

Soporta múltiples plataformas. Intel,® RISC* y Macintosh®* con el mismo código fuente.

Action
(91) 364 02 34

Dany Soft
(91) 654 62 98

Market Software
(91) 593 48 66

Micro Mailers
(93) 280 18 18

Rambla Informática
(93) 540 29 82

Si desea obtener más información sobre Microsoft Visual C ++ 4.0 ó cualquier otr de nuestras herramientas Visuales de desarrollo, ponemos a su disposición un servicio automático de información por F en el número (91) 804 00 96. Puede solicitar información adicional sobre Visu C ++ (código 4007), Visual SourceSafe (4013), Visual Test (4014) y MSDN (600 o visítenos en Internet: **http://www.microsoft.com/devonly**.

*Las versiones de Visual C++ 4.0 para RISC Macintosh se encuentran disponibles por separado.

Microsoft
¿HASTA DONDE QUIERES LLEGAR HOY?